Output-based mesh adaptation using geometric multi-grid for error estimation

Mateusz Gugała

School of Engineering and Materials Science

Queen Mary University of London

Thesis submitted for the degree of Doctor of Philosophy

Supervisor:

Dr. Jens-Dominik Müller

March 26, 2019

Statement of Originality

I, Mateusz Gugała, confirm that the research included within this thesis is my own work or that, where it has been carried out in collaboration with, or supported by others, that this is duly acknowledged and my contribution indicated. Previously published material is also acknowledged.

I attest that I have exercised reasonable care to ensure that the work is original, and does not to the best of my knowledge break any UK law, infringe any third party's copyright or other Intellectual Property Right, or contain any confidential material.

I accept that the College has the right to use plagiarism detection software to check the electronic version of the thesis.

I confirm that this thesis has not been previously submitted for the award of a degree by this or any other university.

Signature: Mateusz Gugała

Date: March 26, 2019

Abstract

The adjoint method in computational fluid dynamics (CFD) made shape optimisation affordable. However, the typical cost of the process is still at least an order of magnitude higher than obtaining a flow solution only. In this work, the author presents methods that help to further reduce the computational effort in optimisation. The first method involves reducing the run-time of the flow solver; the second involves developing a low-cost error estimate that could be used to create a computationally less expensive grid without affecting the accuracy of an objective function.

Implicit solvers are well-established in CFD, but their performance is often limited by the instabilities that arise in the initial convergence stage of the code. To address this issue, a methodology to stabilise an implicit solver using adaptive CFL number adjustment technique is implemented in the in-house code STAMPS. The CFL number is altered at each solver iteration based on the outcome of a line-search algorithm - the Armijo rule. It is shown that the building blocks of a line-search algorithm can be accurately and easily evaluated using automatic differentiation of the Tapenade source code transformation tool without a need to approximate derivatives of discrete system of flow equations. The line-search algorithm is also used to control re-evaluation of Jacobian/preconditioner between solver iterations, by detecting when the linear convergence regime was reached, and the spectra of system matrix eigenvalues are contractive. This work shows that the proposed combination of automatic CFL adjustment and system matrix re-evaluation control result in improvements in solver stability and reductions of the overall run-time of the code.

A method of manufactured solution is used by the author for verification of the discretisation accuracy of the STAMPS solver, as well as for the development of local error estimation. The truncation error, which is defined as a difference between the continuous PDEs and its discrete approximation, can be evaluated exactly using a known manufactured solution and used for verification of error estimation methodology. In this work, a novel low-cost method is presented that estimates the truncation error using building blocks of the geometric multi-grid solver. The methodology requires little implementation effort and uses the same set of multi-grid meshes as the solver. It is shown that a reasonable indication of high-error regions can be achieved, even though the coarse and fine meshes are topologically inconsistent. Although the truncation error can be directly used to obtain an adaptation sensor it is beneficial to apply adjoint-weighting beforehand. The adjoint-weighting of the local truncation error gives an output-based sensor that determines the effect of the local error on the objective function of interest. The output-based sensor can be effectively used for the goal-driven mesh adaptation/coarsening process. This work presents example applications of mesh refinement driven by output-based sensor and mesh regeneration technique.

Acknowledgements

I would like to thank many people that have directly or indirectly contributed to the progress of my PhD work.

I would like to thank my supervisor, Dr. Jens-Dominik Müller, for the opportunity to participate in the AboutFlow[1] research project, and for all the support provided, for which I am very grateful. Thank you for making it possible for me to explore various subjects in my research work, and for the valuable comments that helped me find a link between them.

I was very fortunate to meet many interesting people at QMUL including Jan, Shenren, Orest, Kumar, Oluwadamilare, Yang, Gursharanjit, Siamak, and Xingchen. It was always a pleasure to have coffee and engaging discussions with all of you. A special thank you goes to Jan for being such a supportive friend. I could always count on your help and good advice in and out of the office. Big thanks also to Shenren for many inspiring discussions and for the motivation I received from you.

Thank you to all Early Stage Researchers from AboutFlow and IODA[2] for a friendly atmosphere during projects' meetings and workshops. It was a pleasure to meet you all and exchange experiences while working towards our PhDs.

I am grateful for the support I received during my secondments at Warsaw University of Technology and at Rolls-Royce Germany. Thank you to Professor Jacek Szumbarski, Dr. Marcus Meyer, and Dr. Paolo Adami. Thank you also to PhD students from Warsaw and Rolls-Royce - Łukasz, Wojciech, Georgios, and Ilias - for guiding me around and helping with technical matters.

Thank you to my better half, Olga, for being an amazing wife and friend. No words can express my gratitude for the support, understanding, and love I receive from you. It has been a privilege to share all the good moments with you, and a comfort to get your encouragement and help when needed.

During these years, I always enjoyed going back to my hometown Kozienice and spending time with my amazing family. Whatever I do, I always try to make my parents proud; so this is to say thank you for all your efforts in raising me

[1] http://aboutflow.sems.qmul.ac.uk
[2] http://ioda.sems.qmul.ac.uk

and helping me make right decisions. Thank you also to all my friends from Kozienice and Warsaw. It is impossible to count all the great moments we have had together — I will always hold them in my memory.

I am grateful for being able to use Queen Mary's MidPlus computational facilities for this research, supported by QMUL Research-IT and funded by EPSRC grant EP/K000128/1.

I also gratefully acknowledge the permission of Rolls-Royce Deutschland to publish this work.

Finally, I acknowledge that this work has been conducted within the AboutFlow research project, funded by the European Union FP7 Programme for research, technological development and demonstration under Grant Agreement No. 317006.

Contents

List of Figures

List of Tables

Nomenclature

Indices / lists

i	Node/general purpose index
i_b	Boundary node index
i_e	Edge index (alternatively ij)
i_{pe}	Periodic edge index
ij	Edge index (alternatively i_e)
i_f	Face index
i_{bf}	Boundary face index
i_c	Cell index of primal mesh
N_n	List of nodes
$N_n(i_c)$	List of nodes forming cell i_c
N_e	List of edges
$N_e(i)$	List of edges connected to node i
N_f	List of faces
$N_f(i_c)$	List of faces forming cell i_c
$N_f(ij)$	List of elementary faces constructed around edge ij
N_b	List of boundary patches
$N_b(i)$	List of boundary patches coinciding at node i
N_{bf}	List of elementary boundary faces
$N_{bf}(i)$	List of boundary faces coinciding at node i
$N_{bf}(i_b)$	List of boundary faces coinciding at boundary node i_b
N_c	List of cells
$N_c(i)$	List of cells coinciding at node i

Flow solver

$U, \vec{U}$	Flow variables vector $[\rho, u, v, w, p, \widehat{\nu}]^T$
$W, \vec{W}$	Conservative variables vector $[\rho, \rho u, \rho v, \rho w, \rho e, \widehat{\nu}]^T$
$\vec{f}_c$	Convective flux (functions) vector
$\vec{F}_c$	Convective flux (integrals) vector
$\vec{f}_v$	Viscous flux (functions) vector
$\vec{F}_v$	Viscous flux (integrals) vector
$\vec{q}$	Source term (functions) vector
$\vec{Q}$	Source term (integrals) vector
$R, \vec{R}$	Residual vector
Ω	Control volume
$\partial\Omega$	Boundary that enclose the control volume
S	Surface of control volume
$X, \vec{X}$	Coordinates vector $[x, y, z]^T$
$\vec{n}$	Normal vector of control volume boundary surface $[n_x, n_y, n_z]^T$
$\vec{t}$	Tangent/edge unit vector $[t_x, t_y, t_z]^T$
$\vec{r}_{ij}$	Edge vector
$\vec{s}_{ij}$	Edge (combined dual cell flux face) weight
$\vec{s}_{i_b}$	Boundary weight
$\mathcal{M}, \bar{\mathcal{M}}$	General notation for mesh metrics, i.e. $\vec{s}_{ij}, \vec{s}_{i_b}, \Omega$, etc.
t	Time
ρ	Density
u	X-velocity
v	Y-velocity
w	Z-velocity
p	Pressure
T	Temperature
c	Speed of sound
$\vec{U}_v$	Velocity vector $[u, v, w]^T$
V	Contravariant velocity $\vec{U}_v \cdot \vec{n}$
e	Specific energy
i	Specific internal energy

e_k	Specific kinetic energy
h	Specific enthalpy
h_t	Total specific enthalpy ($h_s + e_k$)
$\widehat{\nu}$	Spalart-Allmaras variable
μ, μ_L	Laminar dynamic viscosity
μ_t	Turbulent dynamic viscosity
$\tau_{i,j}$	Stress tensor component, $i, j = (x, y, z)$
$\nabla U, \nabla \vec{U}$	Gradient of flow variables $[\nabla\rho, \nabla u, \nabla v, \nabla w, \nabla p, \nabla\widehat{\nu}]^T$
Ψ	Limiter function
$(\bullet)_\infty$	Freestream quantity right (e.g. $\rho_\infty, \vec{U}_\infty$, etc.)
$(\bullet)_L$	Reconstructed quantity left (e.g. $\rho_L, \vec{U}_L$, etc.)
$(\bullet)_R$	Reconstructed quantity right (e.g. $\rho_R, \vec{U}_R$, etc.)
A_{ROE}	ROE matrix
ϵ	General notation for errors

Dual solvers

α	Design variable (general), e.g. mesh coordinates $\alpha = \vec{X}$
L	Objective function
N_α	Number of design variables
N_L	Number of objective functions
u	Tangent variables $u = \frac{\partial W}{\partial X}$
ψ	Adjoint variables
f	Tangent system right hand side vector $f = -\frac{\partial R}{\partial X}$
g	Adjoint system right hand side vector $g = \frac{\partial L}{\partial W}^T$
$\widetilde{A}$	Exact Jacobian of the system of equations
R_{TAN}	Tangent residual vector
R_{ADJ}	Adjoint residual vector

Implicit solver stabilisation/acceleration

$RBTS$	Residual-based time stepping
AR	Aspect ratio
$CFL\text{-}RMPG$	Automatic CFL adjustment
$A\text{-}CTRL$	Preconditioner re-computation control
MGS	Multigrid start up
FMG	Full multigrid start up
LMS	Low Mach scaling
A	System of equations matrix
NaN	Not a Number
CFL	Courant-Friedrichs-Lewy number
s	Line search step size
Ma	Mach number
Re	Reynolds number
$GMRES$	Generalised minimal residual method
N_{DoF}	Number of degrees of freedom

Adaptation / Multigrid

h_e	Characteristic mesh size
$(\bullet)_h$	Fine grid quantity/operator (e.g. U_h)
$(\bullet)_H$	Coarse grid quantity/operator (e.g. U_H)
$(\bullet)_h^H$	Quantity at the fine grid interpolated from the coarse mesh
$(\bullet)_H^h$	Quantity at the coarse grid interpolated from the fine mesh
$\mathcal{I}_H^h$	Prolongation operator (from coarse to fine mesh)
$\mathcal{I}_h^H$	Restriction operator (from fine to coarse mesh)
$\vec{Q}_{MG}$	Multigrid source term vector
$\widetilde{L}$	Exact objective function
TE	Truncation error
$\widetilde{TE}$	Exact truncation error
TE^{Ω}	Undivided truncation error (control volume residual error)
OE	Output error

DE	Discretisation error
TS	Truncation-error-based sensor
OS	Output-error-based sensor

Chapter 1

Introduction

One of the earliest attempts to solve the system of flow equations using finite difference and the 'computers' available at the time (i.e. persons that carry out the computations) was presented by Lewis Fry Richardson in 1910 and applied to the problem of accurate weather forecasting, see [8]. The meaning of the term computer has changed significantly since then; equally the Computational Fluid Dynamics (CFD) has evolved. In more recent decades, the capability of CFD has progressed from a very simple potential flow analysis to the fully turbulent large scale applications for complete aeroplanes, cars, engines, etc. Figure 1.1 shows an example of such an application where a full-scale racing car has been analysed to investigate its aerodynamic performance, i.e. lift and drag forces exerted on the car, cooling system efficiency, and brakes cooling.

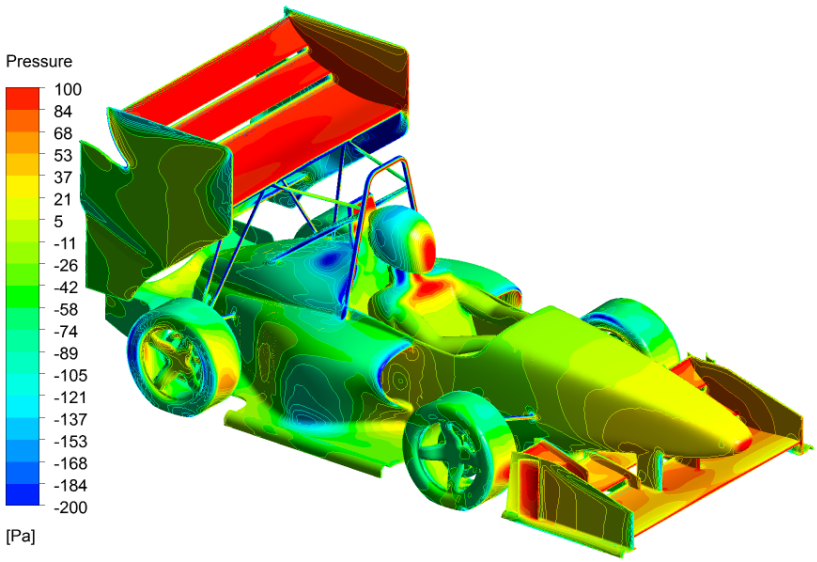


Figure 1.1: Complex CFD analysis of Formula Student vehicle

The advancements in CFD have been possible mainly due to the progress in four key areas (see Jameson [10]) i.e.:

1. Computer Science
2. Mathematics
3. Fluid Mechanics
4. Aeronautical Engineering

Initially, the key driver of the development of CFD was the emerging need for the simulation of a full aircraft and, therefore, the human desire to 'reach the sky'. Creation of first programmable digital computer (ENIAC - Electronic Numerical Integrator And Computer) [11] was an initiation of fast growing computational capabilities of digital machines and parallel computations [12]. Modern Personal-Computers (PCs) can easily deliver 100 GFLOPS, whereas the most powerful supercomputer provides up to 143.5 PFLOPS[3] (Summit, USA). Equally important was a growth of knowledge and creation of new methods in fluid mechanics and mathematics that led to more stable and efficient algorithms, and discretisation schemes.

CFD methods are now a standard tool in analysis and design in the engineering industry. However, in the era of growing computational power and customer demands the flow analysis is no longer a sufficient tool, especially in the aircraft and automotive industry. Numerical optimisation goes beyond the mere analysis of the flow field by modifying control variables such as shape parameters to achieve an optimal flow. There are two main families of CFD optimisation methods: a) stochastic-based [13, 14], and b) gradient-based [15, 16]. The stochastic-based methods (e.g. evolutionary algorithms) are portable and can be easily1y used with various applications and codes. However, they are usually limited to a small number of design variables and are based on the brute force search for the optimum, which requires a large number of objective function evaluations. These methods are not feasible for a large scale optimisation with a rich design space and are not discussed any further here.

[3]https://www.top500.org/lists/top500/, accessed 26 February 2018

Conversely, the optimisation driven by gradient-based methods can provide an optimal solution within acceptable turnaround times. In particular, the use of the adjoint methodology [17] allows a computational cost to be achieved that is independent of the number of design parameters. As a result, a very rich design spaces can be explored, which is of particular importance nowadays as the design of the machines and devices (e.g. engines, aircraft, cars) is already near-optimal. Furthermore, the gradient of an objective function with respect to design variables as well as its intermediate partial derivatives can give an interesting insight into the flow physics and indicate regions of great importance for a particular design case. The strengths of the adjoint method comes at the price of the implementation effort required to obtain derivatives of the primal problem. Nevertheless, the adjoint technique grows in popularity in the CFD field as the advantages outweigh the shortcomings of required development time.

Although the adjoint solution can be used to obtain shape sensitivities at an affordable computational cost, the optimisation process is still much more intensive than the flow solution only. Hence, methodologies are required that allow for further run-time savings, and for the optimisation to be applicable to the wide range of industrial cases. Two methods are presented in this work that allow the computational effort of shape optimisation to be reduced.

The first method is to decrease the run-time of the flow solver. Although the Jacobian-Trained Krylov-Implicit-Runge-Kutta (JT-KIRK) solver shows significant run-time reduction as compared to explicit schemes [18], it still suffers from the initial guess problem [19, 20]. Hence, the initial CFL[4] number has to be often reduced to make sure the stable convergence is obtained. On the other hand, if the CFL number is set to a very low value the advantages of using the implicit solver are lost. A methodology based on line search algorithm that can drive the automatic CFL adjustment is presented in this work along with other convergence acceleration techniques that are described in Chapter 4.

The second method relates to the reduction in mesh size, which could lead to a reduction in computational cost. Adaptive meshing is a method that allows this goal to be achieved while maintaining the accuracy of an objective function

[4]Courant-Friedrichs-Lewy

(i.e. engineering quantity of interest). One of the key requirements of effective mesh adaptation is the right choice of adaptation sensor. The adjoint solution can be used to obtain a robust mesh adaptation sensor, i.e. an output-error-based indicator [29, 30, 25, 31, 32, 33, 34]. An output-based sensor can drive a refinement process that leads to increased accuracy in the estimated objective function of interest at a lower cost and hence allows for a more efficient usage of available computational resources. In other words the computational efficiency, i.e. a ratio of the obtained accuracy of an objective function to a number of computational points, can be increased [35].

In this work, a methodology that allows the efficient estimation of output errors using geometric multigrid solvers [92] is presented. An in-house code STAMPS (Source Transformation Adjoint for Multi-Physics Simulation) developed at Queen Mary University of London (QMUL) is used as a workhorse solver. Several code enhancements have been introduced in STAMPS to achieve an accurate error estimation and extend solver capabilities to a wider range of CFD applications. Furthermore, the stability of implicit solvers linked with an initial guess problem is addressed for nonlinear PDEs. The methodology for automatic CFL adjustment and Jacobian/preconditioner re-computation control is introduced.

1.1 STAMPS

STAMPS is an in-house code developed at QMUL. It originated from a 2D unstructured, geometric multigrid flow solver [93] and was further developed under projects funded by the European Commission, FlowHead[5], AboutFlow[6] and the current IODA[7]. It is a workhorse code for the CFD research work at QMUL. Among STAMPS's primary capabilities one can find:

- flow solver
- adjoint solver

[5]http://flowhead.sems.qmul.ac.uk/
[6]http://aboutflow.sems.qmul.ac.uk/
[7]http://ioda.sems.qmul.ac.uk/

- tangent-linear solver
- shape sensitivity calculation
- truncation error and output error estimation
- mesh deformation
- mesh adaptation

The current and future plans of the software development for STAMPS are focused on MPI[8] parallelisation of the flow and adjoint solvers, and multi-physics applications with the use of adjoint-based optimisation. The ultimate goal is to create a robust multi-disciplinary optimisation platform for engineering applications.

STAMPS is written in Fortran 90/95 programming language [94, 95]. It supports a *GMSH* [96] mesh format for input and output, and an *XDMF*[9] format for the output where the 'heavy data' is stored using *HDF5*[10]. It uses *JSON*[11] open-standard format for solver settings definition, and case setup. The build process of the code is performed via Makefile[12] [97].

STAMPS employs a 2^{nd}-order finite volume discretisation scheme with ROE flux [36] for Reynolds-Averaged Navier-Stokes (RANS) equations (Eq. A.2). It uses vertex-centred approach, i.e. variables stored at mesh nodes, and edge-based numerical integration with dual cells. A geometric multigrid is used for solver acceleration and error estimation, where the sequence of multigrid meshes is generated using tool *hip* [37]. JT-KIRK is a primary flow and adjoint solver in STAMPS [18]. Two mesh deformation techniques are available for altering mesh when the shape optimisation or r-refinement [38] mesh adaptation is considered: a) linear elasticity based [98], and b) spring analogy [39]. Discrete tangent and adjoint solvers are developed in STAMPS using automatic differentiation via source code transformation with the Tapenade AD tool [40, 99].

[8] Message Passing Interface
[9] http://www.xdmf.org/index.php/Main_Page
[10] https://support.hdfgroup.org/HDF5/
[11] https://github.com/josephalevin/fson
[12] https://www.gnu.org/gnu/gnu.en.html

This research involved numerous steps to improve and develop the STAMPS code to make it more accurate, make the implicit solver more stable, and create new capabilities such as an error estimation or linearly exact interpolation operators. These steps included introducing the git[13] source code management system to keep track of the development and maintenance of STAMPS. What is more, the Python-based regression testing was created and many debugging actions performed to improve the code as a joint work with a colleague Jan Hückelheim. The current code passes the test of the debugging and profiling tool Valgrind[14], and results in no errors such as memory leaks or uninitialised variables. A comprehensive framework for solver verification based on a method of manufactured solution [41] was implemented. Undertaking these steps allowed the achievement of a more reliable code and will be beneficial for current and future research work at QMUL and by other related groups.

1.2 Implicit solver stabilisation and enhancements

An advantage of implicit solvers is the possibility for extremely large (infinite) pseudo time step sizes for iterative schemes that can lead to a rapid convergence of the solution to the stationary point [100, 101], e.g. for the RANS system of equations. In particular, Newton-type solvers are known to provide a very rapid (quadratic) convergence when the initial guess is close to a stationary point (converged solution). However, if the initial guess is poor, for example close to critical point where the gradient is near zero, the Newton-type solver will produce a very poor update step [21, 19, 20]. The stability issues of a Newton-type solver related to the choice of the initial guess can be mitigated by reducing speed at which the solution evolves using a finite size pseudo time step. The reduction of pseudo time step can be achieved by adjusting a CFL number [3]. This often requires a user interaction and a tedious manual process of searching for the appropriate CFL number. As a remedy, an algorithm to stabilise and safeguard the implicit solver for the RANS system of equations is introduced in this work. The first step is to analyse the stability of the scheme based on the

[13] https://git-scm.com/
[14] http://valgrind.org/

line-search algorithm [103, 20] known from the optimisation field. Depending on the outcome of the analysis the CFL number is then increased or decreased for the next solver iteration. In STAMPS, the line search algorithm can be implemented very efficiently in the solver thanks to the availability of derivatives; in particular, the forward differentiated residual subroutine that can provide a matrix-vector product required for the line search algorithm - second Wolfe condition [103].

The advantages of the automatic CFL adjustment methodology and Jacobian/preconditioner computation control is discussed in this work. Additionally, other solver enhancements like residual-based time stepping or low Mach scaling are analysed. A study of the implemented techniques was performed on a range of applications.

1.3 Mesh adaptation

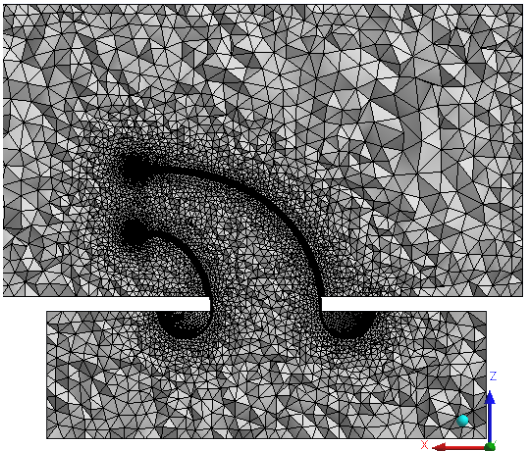
The computational power available nowadays enables the performance of complex CFD simulations with very detailed models, where mesh sizes with more than 10 million computational points (e.g. control volumes for finite volume solvers) are not uncommon, especially in industrial design practice. Despite the growing capabilities of computers, efficient usage of available computational resources is an important consideration, as it allows the CPU cost of a simulation to be controlled. Furthermore, there are still many CFD applications where the computational requirements exceed the feasible cost of everyday usage. An example of such an application comes from the turbo-machinery field, where the accurate estimation of the temperature distribution on a combustion chamber wall requires use of a LES[15] model with chemical reactions of air and fuel (multi-phase flow) and a combustion process. Due to using the LES model, the mesh requirements can quickly rise to more than 100 million computational points [104].

An appropriate distribution of computational nodes within a meshed domain leads to an increased estimation accuracy of an engineering quantity of interest at a lower cost. With the development of CFD many, best practices were created to achieve this goal. Among many general-purpose meshing guidelines one can

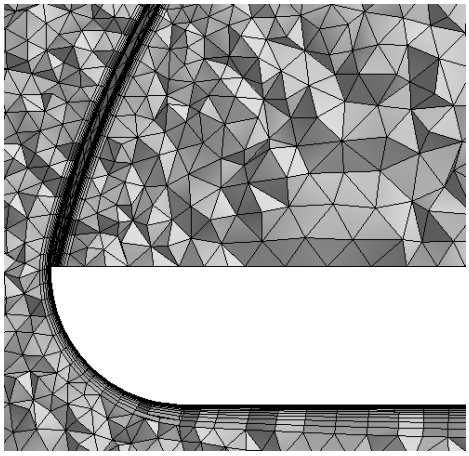
[15]Large Eddy Simulation

distinguish, for example:

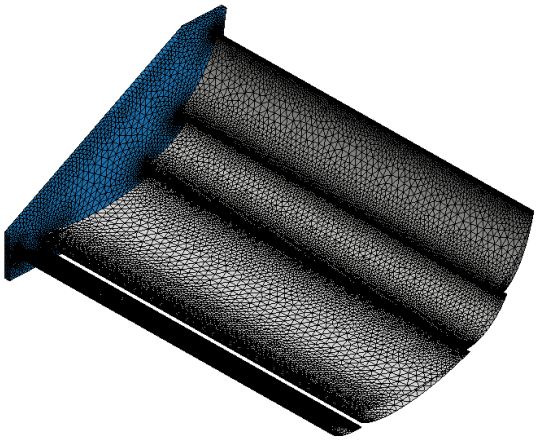
- surface-based and proximity-based refinement (Figure 1.2a), e.g. generation of a fine mesh near walls where the engineering quantity of interest is to be estimated and coarser towards the interior of the computational domain
- directional refinement near the walls (normal wall direction) to capture the physics of a boundary layer (Figure 1.2b)
- curvature-based refinement, e.g. generation of a finer mesh at the wing leading edge (Figure 1.2c)
- mesh refinement in the regions of the computational domain where shock waves are expected for transonic and supersonic flows (Figure 1.2d)
- mesh refinement in the regions where the flow separation is expected e.g. using the so-called 'body of influence' available in ANSYS [105].



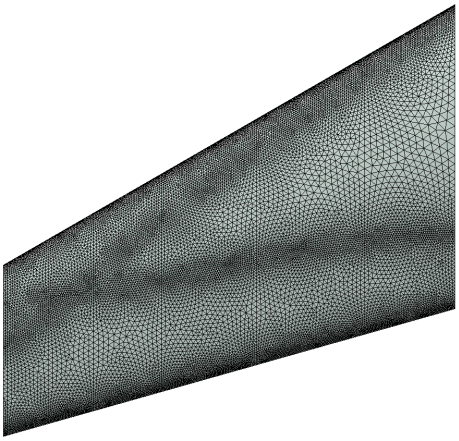
(a) Surface-based refinement - hovercraft nozzle



(b) Directional refinement - boundary layer near the wall



(c) Curvature-based and surface-proximity refinement - wing of a race car



(d) Refinement for the lambda-shaped shock - M6 wing [106]

Figure 1.2: Examples of best practices in mesh generation

These techniques are available in most modern meshing and CFD simulation tools like ANSYS Meshing [105], STAR CCM+ [107], Altair HyperMesh [108], or OpenFoam [109]. Examples can also be found of meshing best practices for specific applications, e.g. automotive external aerodynamics [110], marine applications [111], or urban environment simulations [112]. Although these rough guidelines help to accurately resolve the shape (geometry) of analysed object capturing regions of high curvature, tight gaps etc, they are disconnected from physical model used in a simulation. The influence of boundary conditions, presence of shocks, or engineering quantity that is of interest is not predicted by the current meshing tools. Hence, a method is required that can indicate where the mesh should be refined and where the coarse cells are sufficient.

The main goal of mesh adaptation is to provide an optimal distribution of

the computational points in the discretised space (mesh). The first challenge is to estimate errors due to the discrete approximation of the continuous model (e.g. systems of flow equations). In CFD, the most popular are *a posteriori* error estimation methods that rely on the discrete solution of the PDEs and allow the actual errors for a given computational grid to be derived [113, 114, 42]. However, the *a priori* methods that are based on the mathematical analysis of PDEs can also provide valuable information [6]. Some *a priori* error analysis was also presented in [7, 43]. In this work, however, the *a posteriori* techniques are employed, and the *a priori* approach will not be discussed further as it is outside the scope of this thesis.

Various types of error estimation have been published in the literature. The most popular ones are: a) solution errors DE, b) truncation errors TE, and c) output errors OE [44]. As described in Section 5.2 these errors are closely related to each other. The thesis focuses on the output errors as they allow a recognition of regions within a computational domain where not only the solution/truncation errors are high but also the errors have a large influence on an engineering quantity of interest, e.g. lift, drag, or pressure loss. This is of particular importance as the primary goal of the engineering CFD simulations is to get an accurate estimate of the cost function of interest.

Finally, an effective adaptation method that uses a form-of-estimated-error (adaptation sensor) is required to perform the actual modification of the original grid. The most popular methods include: a) r-refinement [45] - relocation of nodes without changing mesh size, b) h-refinement [46, 31] - subdivision of original mesh cells, c) p-refinement [47, 48] - adjusting order of interpolation polynomial, d) re-meshing [43, 49] - rebuilding the original mesh. This work uses a re-meshing methodology because it allows exploiting a well-developed and robust meshing tools and libraries that can consistently generate high quality computational grids with a good control over the refinement and coarsening regions.

Two meshing tools are used in this work:

- BoxerMesh[16] [86] - commercial meshing tools developed by Cambridge Flow Solutions.
- Mmg3D[17] [88] - a robust, open-source and multi-disciplinary software for re-meshing developed by Mmg open source Consortium.

A more extensive overview of error estimation methods and adaptation techniques is provided in the introduction to Chapter 5.

1.4 Outline and contributions

This work is organised in the following fashion. First, the QMUL in-house code STAMPS is introduced in Chapter 2. A full description of the flow solver, discretisation scheme, and boundary conditions is presented. The improvements made by the research towards the stability and accuracy of the code are highlighted, including the implementation of a differentiable slope limiter and formulation of a linearly transparent gradient computation method.

Chapter 3 presents the dual solvers available in STAMPS, i.e. tangent and adjoint, and an efficient way to calculate shape sensitivities for gradient-based optimisation with the use of the algorithmic differentiation (AD) tool Tapenade [40, 99]. The mathematical derivation, as well as implementation details, are provided along with a range of application examples.

An improved efficiency of the JT-KIRK [18] implicit solver and various convergence enhancement techniques are studied in Chapter 4. In this work, an extension of the automatic CFL adjustment driven by line search algorithm that was proposed by Michalak and Olivier-Gooch [20] for compressible Euler solver, and by Pawlowski et al. [22] or Tuminaro et al. [23] for incompressible solvers is presented. The author's own contributions are related to application of the automatic CFL adjustment technique to the compressible turbulent solver (STAMPS), and an exact evaluation of derivatives in the line search algorithm (using Tapenade algorithmic differentiation tool). All implicit solver enhancements such as

[16] http://www.cambridgeflowsolutions.com/en/products/boxer-mesh/
[17] https://www.mmgtools.org/mmg-remesher-downloads

automatic CFL adjustment, Jacobian re-computation control, solution recovery mechanism, or low Mach number scaling are implemented in STAMPS code and tested on a set of cases (inviscid and viscous).

Chapter 5 presents an efficient procedure for truncation error estimation that exploits the building blocks of a geometric multigrid solver. The methodology is tested using the method of manufactured solution (Appendix A.5) and a cube computational domain. An effective output-based adaptation sensor is then obtained through the adjoint-weighting of the estimated truncation error and used for driving a mesh adaptation process (re-meshing).

The truncation error estimation using geometric multigrid solver and topologically inconsistent grids is the original contribution of this work. The estimation is carried out on a coarse grid and then interpolated to the finest mesh. This method is computationally inexpensive and can be easily implemented in any geometric multigrid solver. In this work the method was explored on two separate CFD codes; the QMUL in-house solver STAMPS, and the Rolls-Royce proprietary CFD code Hydra. Three example application cases of output-based refinement are presented which are also author's original contribution.

1. 3D cube with a manufactured solution - re-meshing refinement using BoxerMesh and Hydra's flow and adjoint solvers.

2. Subsonic Onera M6 wing [106] - re-meshing refinement using mmg3d and STAMPS's flow and adjoint solvers.

3. Transonic Onera M6 wing - same as point 2.

Finally, conclusions and suggestions for future work are provided in Chapter 6.

Chapter 2

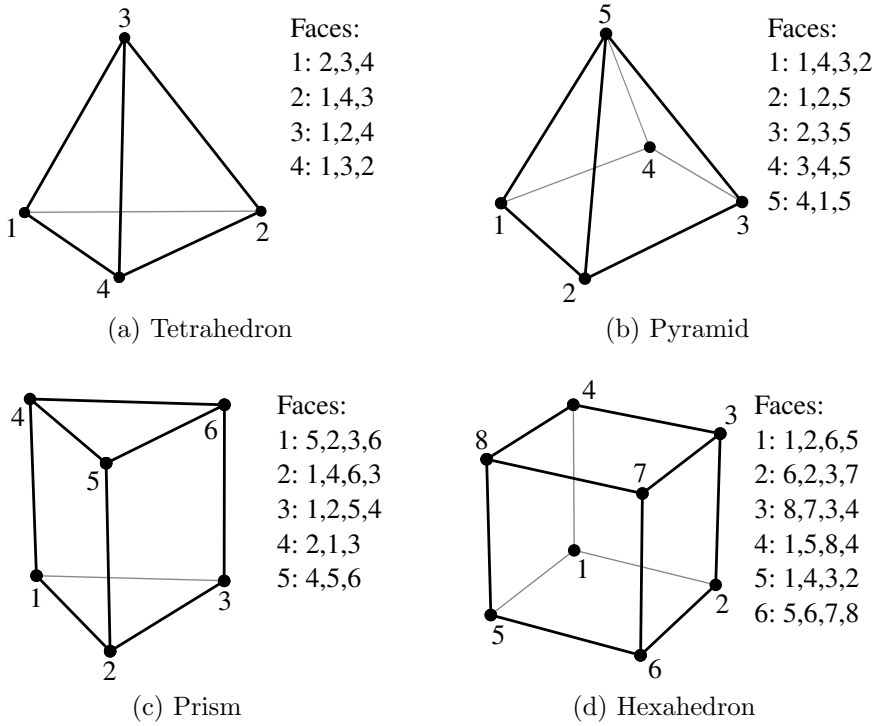
STAMPS primal (flow) solver

This chapter presents the Reynolds–Averaged Navier–Stokes system of equations (RANS) and the numerical method used to discretise and solve. Chapter 2 also discusses a detailed list of primary solver capabilities and improvements implemented by the author. The Chapter concludes with the verification and validation of the flow solver.

2.1 Introduction

STAMPS solves 3D compressible RANS equations on unstructured grids. It employs finite-volume, vertex-centred discretisation with an edge-based data structure. Although this is a standard technique in the CFD field, the goal is to demonstrate how the AD adjoints can efficiently be developed with typical CFD codes. It is important to note that the adjoint method can also apply to other discretisaiton methods, e.g. higher-order schemes.

Among the supported element types, there are triangular and quadrilateral boundary cells, and basic 3D element types as presented in Figure 2.1. The cell numbering and face lists are also provided in the figure.



(a) Tetrahedron (b) Pyramid

(c) Prism (d) Hexahedron

Figure 2.1: 3D element types supported in STAMPS and face numbering

As STAMPS uses node-centred discretisaiton approach where variables are stored at nodes of the original mesh, a new computational grid (dual mesh) has to be constructed. The so-called *median-dual* volumes, or simply dual volumes, are constructed around nodes of the original grid. A 2D example of a dual cell is shown in Figure 2.2. In this work the notation Ω refers to the volume of the dual cell unless stated otherwise.

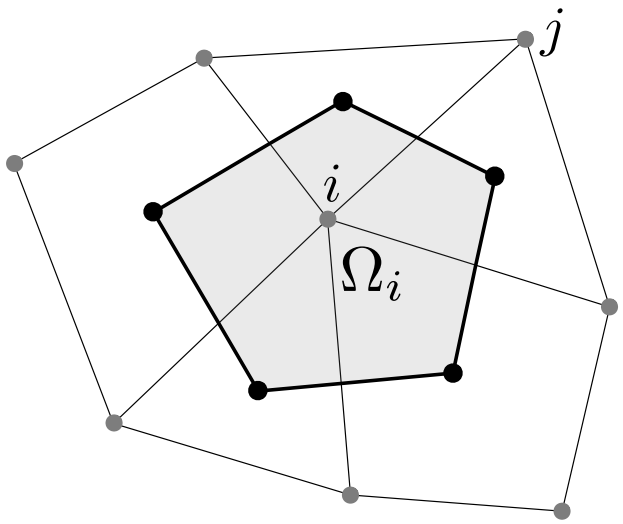


Figure 2.2: 2D Dual control volume (dual cell)

The graphical representation of the dual grid is presented in Figure 2.3.

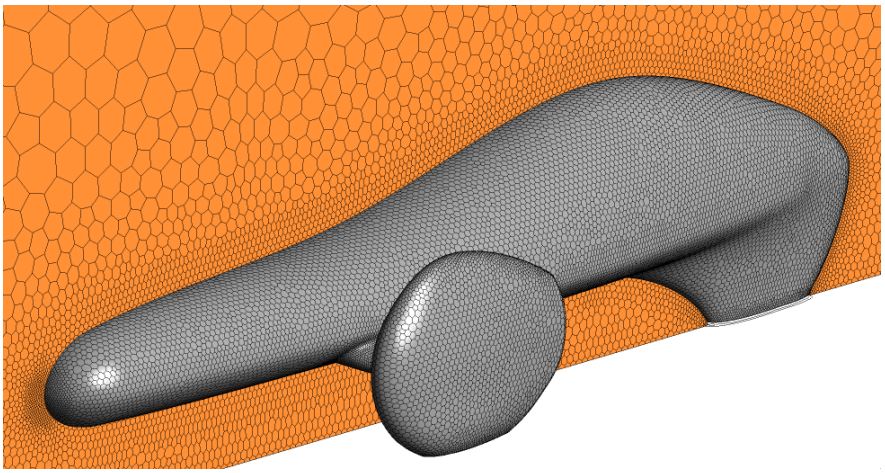


Figure 2.3: 3D example of dual mesh (polyhedral) constructed based on a tet mesh - fuel efficient vehicle

The summary of the key flow solver information and capabilities is presented below. The appropriate reference and links to sections that contain detailed descriptions on each solver feature are provided. The $(Default)$ tag indicates the recommended setting in STAMPS when more than one option is available.

Flow physics

- Inviscid
- Laminar
- Turbulent (SA[18])
- Steady / unsteady

Discretisation - Section 2.3

- Vertex-centred, finite-volume with dual mesh. 2^{nd}-order spatial discretisation for a general tet and smooth hex and mixed grids, see Section 2.7
- Edge-based data structure. Boundary weights stored at the boundary nodes
- Temporal discretisation for unsteady flows: implicit dual time-stepping using backward differentiation formula 2 (BDF2) [66]

[18]Spalart-Allmaras

Solution reconstruction - Section 2.3.2

- 1^{st}-order
- 2^{nd}-order gradient-based

Gradient calculation using Green-Gauss - Section 2.3.3

- Standard edge-based
- Cell-based, consistently accurate for any mesh type $(Default)$

Limiters - Section 2.3.6

- (BJ): Barth-Jespersen [121]
- (VE): differentiable modified Venkatakrishnan (VE) [51, 53]
- (VEM): modified VE version [54] $(Default)$

Boundary conditions - Section 2.3.10

- Wall (adiabatic): inviscid/viscous
- Far-field
- Subsonic inlet/outlet
- Symmetry
- Periodic (rotation/translation)

Flux calculation - Sections 2.3.7-2.3.8

- ROE scheme [36] $(Default)$
- AUSM^{up}_{+} scheme [50]

Solvers - Section 2.4

- Explicit with local time-stepping [65]
- Implicit block-Jacobi. It uses a preconditioner based on the approximate block-diagonal Jacobian of the 1^{st}-order spatial discretisation scheme, hence the name block-diagonal Jacobian. A term 0^{th}-order Jacobian or simply block Jacobian will be used in this work as an alternative name for the block-diagonal Jacobian.
- Implicit JT-KIRK [18] with GMRES [68] linear solver. It uses a preconditioner based on the exact Jacobian of the 1^{st}-order spatial discretisation scheme, in short, the 1^{st}-order Jacobian. A $(Default)$

Geometric multigrid convergence acceleration - Section 2.6

- Standard V-cycle [92] available with any STAMPS's solver
- Multigrid start-up for solution initialisation
- 2^{nd}-order solution prolongation operator and 1^{st}-order residual restriction to transfer information between coarse and fine meshes - operators obtained using a minimum-norm solution. 2^{nd}-order accurate solution restriction available using a gradient-based approach
- Coarse grids generated using an external tool *hip* that exploits an edge-collapsing algorithm [37]

Error estimation - Section 5.2

- Truncation error estimation using geometric multigrid [87]
- Output error estimation (adjoint-weighted truncation error).

2.2 Mathematical model

The governing system of compressible RANS equations can be written using continuous integral form for a stationary control volume Ω with boundary $\partial\Omega$

as presented in Eq. 2.1. $\vec{W}$ is a vector of conservative variables, $\vec{f_c}$ is vector of convective fluxes, $\vec{f_v}$ is a vector of viscous fluxes, and $\vec{q}$ is a vector of source terms.

$$\frac{\partial}{\partial t}\int_{\Omega}\vec{W}d\Omega + \oint_{\partial\Omega}(\vec{f_c} - \vec{f_v})\,dS = \int_{\Omega}\vec{q}\,d\Omega \tag{2.1}$$

The vector of conservative variables $\vec{W}$ consists of six components

$$\vec{W} = \begin{bmatrix} \rho \\ \rho u \\ \rho v \\ \rho w \\ \rho e \\ \widehat{\nu} \end{bmatrix} \begin{array}{l} density \\ x\text{-}momentum \\ y\text{-}momentum \\ z\text{-}momentum \\ energy \\ modified\ eddy\ viscosity\ (SA\ variable) \end{array} \tag{2.2}$$

The convective flux vector $\vec{f_c}$ is presented in Eq. 2.3. The variable p stands for pressure and the velocity components $[u, v, w]^T$ form the velocity vector $\vec{U_v}$ (subscript v is used to avoid confusion with vector of flow variables $\vec{U}$).

$$\vec{f_c} = \begin{bmatrix} \rho V \\ \rho u V + n_x p \\ \rho v V + n_y p \\ \rho w V + n_z p \\ \rho e V + p V \\ \widehat{\nu} V \end{bmatrix} \tag{2.3}$$

The variable $V = \vec{U_v} \cdot \vec{n} = n_x u + n_y v + n_z w$ is the velocity normal to the surface element dS. The surface dS is defined by the unit normal vector $\vec{n} = [n_x, n_y, n_z]^T$.

The viscous flux vector $\vec{f_v}$ is presented in Eq. 2.4. Both fluxes are integrated over a closed surface S of a control volume Ω. The matrix of viscous stresses - τ_{ij}, work done by shear stresses and terms related to thermal conductivity - Θ_i,

and other quantities are described in Appendix A.2.

$$\vec{f_v} = \begin{bmatrix} 0 \\ n_x\tau_{xx} + n_z\tau_{xy} + n_z\tau_{xz} \\ n_x\tau_{yx} + n_z\tau_{yy} + n_z\tau_{yz} \\ n_x\tau_{zx} + n_z\tau_{zy} + n_z\tau_{zz} \\ n_x\Theta_x + n_z\Theta_y + n_z\Theta_z \\ \frac{1}{\sigma}\left(\nu_L + \widehat{\nu}\right)\left(\nabla\nu \cdot \vec{n}\right)\widehat{\nu} \end{bmatrix} \tag{2.4}$$

The source term vector q (Eq. 2.5)has all components zero except for the last which is a volume source for the Spalart-Allmaras (SA) equation defined in Appendix A.2 - Eq. A.14.

$$\vec{q} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ SA_{src} \end{bmatrix} \tag{2.5}$$

The Spalart-Allmaras turbulence model [55, 117] and the concept of eddy viscosity is used to close the system of RANS equations. The NASA Langley Research Centre website[19] (on the turbulence modelling) provides a useful guide to the SA model and its various modifications. STAMPS uses the version of SA model with a modified vorticity. No limiting is used for turbulent viscosity [117].

Additionally, because it will often be used in this work, it is useful to define

[19]http://turbmodels.larc.nasa.gov/index.html

the vector of flow/primitive variables $\vec{U}$ - Eq. 2.6.

$$\vec{U} = \begin{bmatrix} \rho \\ u \\ v \\ w \\ p \\ \widehat{\nu} \end{bmatrix} \begin{matrix} density \\ x\text{-}velocity \\ y\text{-}velocity \\ z\text{-}velocity \\ pressure \\ Spalart\text{-}Allmaras\, variable \end{matrix} \tag{2.6}$$

The compact integral form of flow equations shown in Eq. 2.1 is presented in its expanded form in Appendices A.1 and A.2. All physical constants and reference quantities are also discussed.

The system of Eqs. 2.1 is discretised in STAMPS using the so-called *method of lines*, that is, using separate discretisation in space and time [5]. Details are described in the following Sections 2.3-2.4.

2.3 Spatial discretisation

The residual vector that represents spatial integrals of Eq. 2.1 is introduced in Eq. 2.7.

$$\vec{R}(\vec{W}) = \oint_{\partial\Omega_i} \vec{f_c}\, dS_i - \oint_{\partial\Omega_i} \vec{f_v}\, dS_i - \int_{\Omega_i} \vec{q}\, d\Omega_i \tag{2.7}$$

The continuous residual vector $\vec{R}$ can be approximated using the discrete residual vector $\vec{R}_i$ as presented in a general form in Eq. 2.8. Vectors $\vec{F}_{c,i}$, $\vec{F}_{v,i}$, and $\vec{Q}_i$ correspond to the discretised integrals of the continuous mathematical model 2.7, and the index i refers to discrete computational point.

$$\vec{R}(\vec{W}) \approx \vec{R}_i(\vec{W}) = \vec{F}_{c,i} - \vec{F}_{v,i} - \vec{Q}_i \tag{2.8}$$

This section provides details on the key aspects of spatial discretisation scheme used in STAMPS:

- dual mesh generation and calculation of geometric quantities

- solution reconstruction
- discretisation of each component of the residual $\vec{R}_i$ (flux functions and source terms).

2.3.1 Geometric properties

As a primary step towards the discretisation of flow equations (Eq. 2.1), a computation of geometric quantities such as normals of flux faces and volumes is required. As initially discussed in Section 2.1, STAMPS uses node-centred discretisation scheme (flow variables stored at mesh nodes) which require construction of a dual mesh. The original (base) grid is used to construct a dual mesh, where the former is supplied by the user as an input. This section describes the details of the dual mesh construction, geometrical quantities computation, and creation of appropriate connectivity lists.

An example 2D dual cell (often referred to as median-dual cell) constructed on the original grid is shown in Figure 2.4.

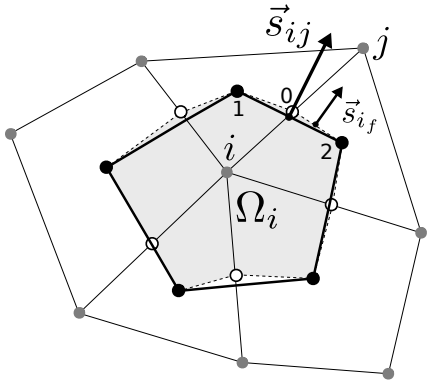


Figure 2.4: Constructions of a 2D dual control volume (dual cell). Dashed line - intermediate construction step of the dual volume, solid line - final dual cell. Hollow circles - edge centres, filled circles - cell centres

The construction starts by calculating geometric centres for edges and cells. Next, the centres are connected as presented by dashed line in Figure 2.4, and individual edge normals $\vec{s}_{i_f}$ are calculated. The normals $\vec{s}_{i_f}$ of example edges 0–1 and 0–2 that coincide at midpoint of edge ij of the original mesh are summed up to obtain the so-called edge weights $\vec{s}_{ij}$ where the name refers to the edge-

based data structure used in the typical vertex-centred discretisation scheme. The resultant edge weights are necessary to perform efficient numerical integration for the vertex-centred discretisation scheme (calculate sum of fluxes for each dual cell and each flow equation).

The final dual cell can be visualised as shown by solid line in Figure 2.4; it represents a polyhedral cell build by connecting cell centres between the coinciding cells of the original mesh. It is important to highlight that the direction of the edge weight vector is aligned with the direction of the normal of edge 1–2 (solid line), whereas the magnitude is equal to total length of construction edges formed with example points 0–1 and 0–2 (dashed lines) shown in Figure 2.4. Although this 'inconsistency' can create some confusions it allows to perform efficient numerical integration of fluxes while maintaining accuracy. The presented median-dual finite volume discretisation is equivalent to the Galerkin finite element scheme with linear elements [91].

A 3D dual cell can be formed accordingly as illustrated in Figure 2.5 where the construction of a single face of a dual cell is presented. The solid black and hollow circles in the figures refer to the cell and face centres of the base mesh, respectively. The hollow square refer to the edge-midpoint.

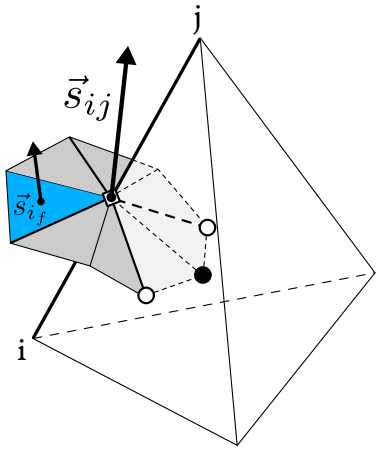


Figure 2.5: Construction of a single flux face for a 3D cell

Geometric quantities

List of all geometrical quantities calculated in the pre-processing step in STAMPS is provided below.

- Edge weights $\vec{s}_{ij}$, constructed around edges using edge, face, and cell centres of the original mesh as shown in Figure 2.5. The final edge weight is calculated using Eq. 2.9, where the $\vec{s}_{i_f}$ is the elementary triangle weight with the magnitude equal to face area and $N_f(ij)$ is a list of elementary triangles constructed around edge ij.

$$\vec{s}_{ij} = \sum_{i_f \in N_f(ij)} \vec{s}_{i_f} \tag{2.9}$$

  The edge weights are obtained in the same way for any of the cell types presented in Figure 2.1. The $\vec{s}_{ij}$ vector direction is defined to point from the lower node index i to the higher node index j. The i and j nodes are accessed using the appropriate connectivity that is obtained in the solver pre-processing step.

- Boundary weights $\vec{s}_{ib}$, required to close the dual volumes at the boundaries. They are constructed using the boundary node i_b as well as edge and face centres - Figure 2.6.

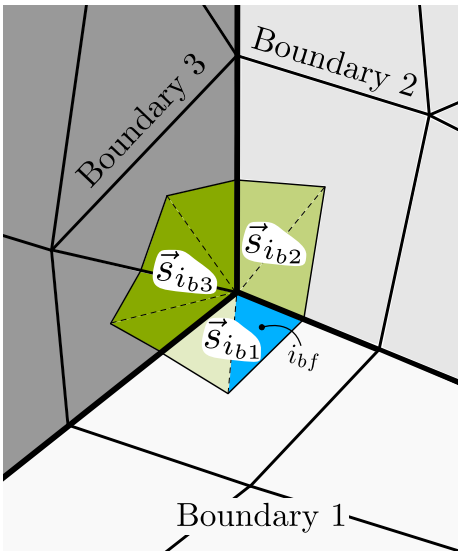


Figure 2.6: Boundary weights definition - separate weights for each boundary patch

  Boundary weights are calculated separately for each boundary patch as

presented in Eq. 2.10, where the $N_{bf}(i_b)$ is a list of elementary triangles i_{bf} constructed at the boundary node i_b, with i_b being shared between boundary patches - Figure 2.6. Storing boundary weights separately for each boundary patch is required for boundary conditions implementation and objective function evaluation.

$$\vec{s}_{ib} = \sum_{i_{bf} \in N_{bf}(i_b)} \vec{s}_{i_{bf}} \tag{2.10}$$

The boundary weight is defined to point outward from the computational domain.

- Dual volumes Ω, calculated using the Green-Gauss approach (Eq. 2.11) by setting the vector field $\vec{F}$ equal to an edge midpoint coordinate $\vec{X}_{ij} = 0.5(\vec{X}_i + \vec{X}_j)$.

$$\iiint_{\Omega} \nabla \cdot \vec{F} d\Omega = \oiint_{S} \vec{F} \cdot \vec{n} dS \tag{2.11}$$

The numerical integration is then performed according to Eq. 2.12. Using the fact that the gradient of field $\vec{X} = [1, 1, 1]^T$, the dual volume can be calculated as shown in Eq. 2.12. A boundary contribution has to be added when considering boundary dual cells i.e. when the list of boundary patches coinciding at node i is not an empty set ($N_b(i) \neq \varnothing$).

$$\Omega_i = \frac{1}{3} \sum_{j \in N_e(i)} \vec{X}_{ij} \cdot \vec{s}_{ij} + \begin{cases} 0, & N_b(i) = \varnothing \\ \frac{1}{3} \sum_{i_b \in N_b(i)} \vec{X}_i \cdot \vec{s}_{i_b}, & N_b(i) \neq \varnothing \end{cases} \tag{2.12}$$

Connectivity lists

In addition to computation of geometric quantities, the following connectivity lists are required for the efficient implementation of an edge-based scheme:

- `edge2nodes` - pointer from edge to list of two forming nodes. It is of size 2 $\times$ number-of-edges. The two nodes forming edge ij are obtained as follows

$$\begin{aligned} i &= \texttt{edge2nodes}(1, ij) \\ j &= \texttt{edge2nodes}(2, ij) \end{aligned} \tag{2.13}$$

where node IDs are stored in the ascending order i.e. $i < j$. This defines the positive direction of the edge weight $\vec{s}_{ij}$ that point from node i to node j.

- `bNode2node` - pointer from boundary node to global node ID. It is of size number-of-boundary-nodes. Note that the `bNode2node`(i_b) can point to the same global node ID for several different boundary node IDs, which is the case for nodes that lies at the boundary patch intersections e.g. the corner node presented in Figure 2.6 is represented by three boundary nodes - separate for each patch. Boundary nodes are numbered contiguously within a patch.

- `bNodeStartID` - pointer to the starting and ending boundary node that form a given boundary patch. The boundary contribution, e.g. in Eq. 2.10 or 2.38, is added using an outer loop over each boundary patch and and inner loop over boundary nodes within the given patch. This approach allow for appropriate treatment of boundary conditions and evaluation of an objective function.

```
do i_patch = 1, n_patches
  ib_start = bNodeStartID(i_patch)
  ib_end   = bNodeStartId(i_patch+1)-1
  do ib = ib_start, ib_end
     i = bNode2node(ib)
     ...
  end do
end do
```

All the pre-processing information is required for the numerical integration of the system of flow equations 2.1.

2.3.2 Solution reconstruction

Vertex-centred solvers store solution vector at mesh nodes; however, the convective and viscous fluxes ($\vec{F}_{c,i}$ and $\vec{F}_{v,i}$) shown in Eq. 2.8, are evaluated at the faces

of the control volume Ω - Figure 2.7. This requires reconstruction of a solution vector at the left (L) and right (R) sides of the flux face.

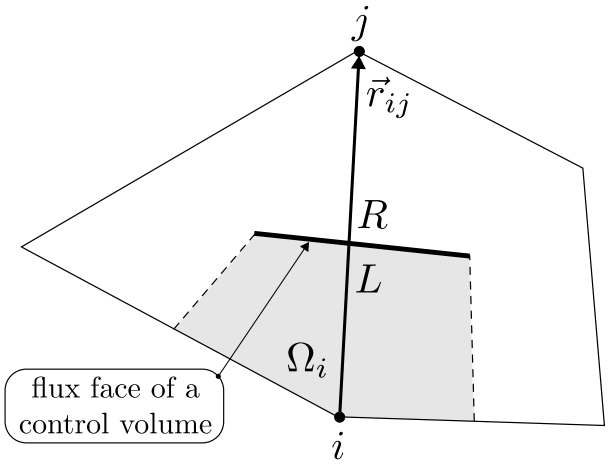


Figure 2.7: Left and right side of a flux face of a dual volume Ω_i - 2D example

Given a scalar field ϕ, gradient $\nabla\phi$, and edge vector $\vec{r}_{ij}$, the flux can be linearly reconstructed using Eq. 2.14, assuming that the method used for gradient calculation is at least first-order accurate. Additionally, the limiter function Ψ (values from 0 to 1) is used to prevent the generation of oscillations and spurious solutions.

$$\begin{aligned} \phi_L &= \phi_i + \frac{1}{2}\Psi_i \left(\nabla\phi_i \cdot \vec{r}_{ij}\right) \\ \phi_R &= \phi_j - \frac{1}{2}\Psi_j \left(\nabla\phi_j \cdot \vec{r}_{ij}\right) \\ \vec{r}_{ij} &= \vec{X}_j - \vec{X}_i \end{aligned} \tag{2.14}$$

Linear reconstruction is one of the requirements for the 2^{nd}-order spatial discretisation scheme. In STAMPS, a 1^{st}-order accurate discretisation is also available. For this case the reconstruction is done simply by setting $\phi_L = \phi_i$ and $\phi_R = \phi_j$.

In the current implementation, the 2^{nd}-order reconstruction (Eq. 2.14) is used for all state variables except for the turbulent variable which is reconstructed using 1^{st}-order scheme. This is due to the solver stability issue with 2^{nd}-order reconstruction for turbulent variable which requires further investigation. However, this is outside the scope of this research and is therefore not discussed further.

2.3.3 Gradient

Gradient computation is required for solution reconstruction (refer Eq. 2.14), and discretisation of viscous fluxes (see Eq. 2.4). One of the requirements for achieving 2^{nd}-order discretisation accuracy is to obtain a gradient that is exact for linear fields i.e. polynomials of degree 1 [62]. In STAMPS, the Green-Gauss theorem (Eq. 2.15) is used for gradient computation.

$$\iiint_{\Omega} \nabla\phi d\Omega = \oiint_{S} \phi\vec{n}\, dS \tag{2.15}$$

The continuous formula for the gradient of a scalar field ϕ at the control volume Ω is presented in Eq. 2.16.

$$\nabla\phi = \frac{1}{\Omega} \oiint_{S} \phi\vec{n}\, dS \tag{2.16}$$

This section presents the typical edge-based approach that provides 2^{nd}-order solution reconstruction only on specific grid types, as well as the accuracy enhancements implemented by the author.

2.3.4 Edge-based gradient

The continuous integral from Eq. 2.16 can be discretised using an edge-based data structure and the numerical integration can be performed as presented in 2.17.

$$\nabla\phi_i = \frac{1}{\Omega_i} \sum_{j \in N_e(i)} \phi_{ij}\vec{s}_{ij} + \begin{cases} 0, & N_b(i) = \varnothing \\ \sum_{i_b \in N_b(i)} \phi_i \vec{s}_{i_b}, & N_b(i) \neq \varnothing \end{cases} \tag{2.17}$$

The gradient at mesh node i is computed using scalar quantity ϕ_{ij} evaluated at the given edge ij using a simple average (Eq. 2.18), and $\vec{s}_{ij}$ is an edge weight - refer Section 2.3, Figure 2.5. Boundary contribution has to be added when considering the boundary dual cell.

$$\phi_{ij} = 0.5(\phi_i + \phi_j) \tag{2.18}$$

The gradient calculated with the presented methodology is exact for linear fields on regular hexahedral meshes (all cell faces square) and interior control volumes

of general tetrahedral grids. The gradient is not exact for linear fields on interior and boundary control volumes of general hexahedral or mixed grids as well as on boundary control volumes of all-tet meshes. However, for the case of all-tet meshes, a simple correction can be performed to obtain linearly transparent gradients at the boundaries [90].

Edge-based gradient with boundary correction

The accuracy of the base edge-based gradient calculation for all-tet grids at the boundaries can be made consistent with accuracy at the interior nodes (exact for linear fields) by introducing an appropriate correction proposed by Barth [90]. The corrected equation is presented in Eq. 2.19.

$$\nabla\phi_i = \frac{1}{\Omega_i}\sum_{j\in N_e(i)}\phi_{ij}\vec{s}_{ij} + \begin{cases} 0, & N_b(i) = \varnothing \\ \sum_{i_{bf}\in N_{bf}(i)}\phi_{i_{bf}}\vec{s}_{i_{bf}}, & N_b(i) \neq \varnothing \end{cases} \tag{2.19}$$

For the original mesh face ijk the scalar quantity $\phi_{i_{bf}}$ at point i_{bf} from Figure 2.8 can be obtained through the linear interpolation as shown in Eq. 2.20. The partial face weight is computed as follows: $\vec{s}_{i_{bf}} = \vec{s}_{ijk}/3$ and the $N_{bf}(i)$ is a list of boundary faces coinciding at node i.

$$\phi_{i_{bf}} = 6\phi_i + \phi_j + \phi_k \tag{2.20}$$

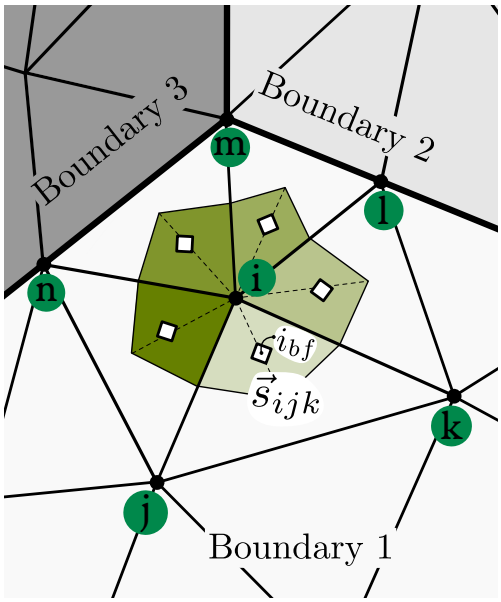
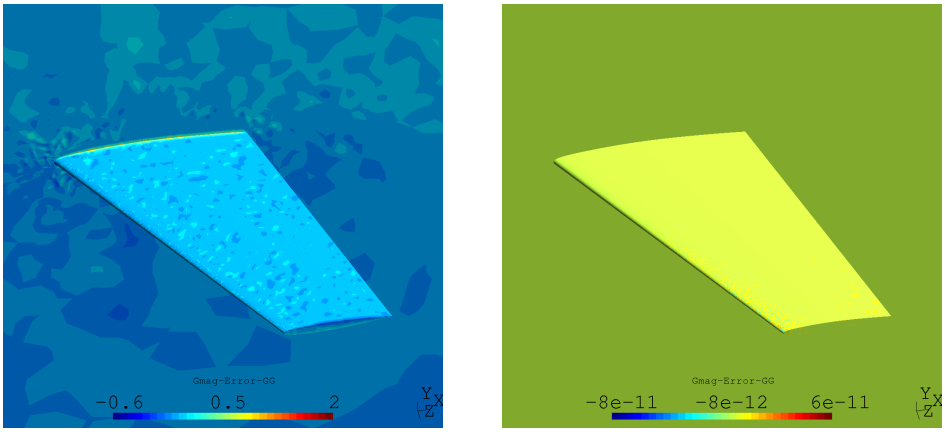


Figure 2.8: Boundary correction for dual volume at i

Figure 2.9 shows the gradient accuracy for the flow over the Onera M6 wing [106] on an all-tet mesh with contours of gradient error for the base - Eq. 2.17 and corrected - Eq. 2.19 gradient computation. The linear field of the form $\phi = x + 2y + 3z$ was imposed, and the gradient computed. The relative error in gradient magnitude is presented on Figure 2.9 - see the colour legends. The L_1 and L_∞ norms are calculated for both gradient calculation methods:

- Base: $L_1 = 3.53 \cdot 10^{-3}$, $L_\infty = 1.96$
- Corrected: $L_1 = 5.34 \cdot 10^{-15}$, $L_\infty = 6.48 \cdot 10^{-13}$

The results confirm the validity of the correction. However, the improved accuracy comes at the cost of storing additional connectivity i.e. pointers from boundary face to nodes and boundary face weights $\vec{s}_{ijk}$, and is exact for linear fields only for all-tet meshes.



(a) Base approach - Eq. 2.17 (b) Boundary correction - Eq. 2.19

Figure 2.9: Gradient magnitude error at the boundaries - linear field test ($\phi = x + 2y + 3z$)

2.3.5 Cell-based gradient

This section proposes a gradient calculation method that is accurate for linear fields on a general mesh type. It is based on an efficient cell-based approach, where the term 'cell' refers to the primary grid cell (original mesh). In this method, the gradient for each primary cell type supported by STAMPS i.e. tetrahedron, pyramid, prism and hexahedron, can be efficiently calculated using a matrix-vector product of pre-computed nodal coefficients C and nodal values of a scalar field ϕ - Eq. 2.21.

$$\nabla\phi_{i_c} = \sum_{i \in N_n(i_c)} \left(\phi_i \vec{C}_i\right) \Big/ \Omega_{i_c} \tag{2.21}$$

The variable $\vec{C}_i$ in Eq. 2.21 is a single column of matrix C for a given node i, and $N_n(i_c)$ is a list of nodes forming a given cell i_c. An example matrix C is shown in Eq. 2.22, which is derived for a hexahedral element (Figure 2.10). In this case, the matrix of coefficient C is of size 3×8 (3 dimensions $\times$ 8 nodes).

$$C = \begin{bmatrix} c_{x,1} & c_{x,2} & c_{x,3} & c_{x,4} & c_{x,5} & c_{x,6} & c_{x,7} & c_{x,8} \\ c_{y,1} & c_{y,2} & c_{y,3} & c_{y,4} & c_{y,5} & c_{y,6} & c_{y,7} & c_{y,8} \\ c_{z,1} & c_{z,2} & c_{z,3} & c_{z,4} & c_{z,5} & c_{z,6} & c_{z,7} & c_{z,8} \end{bmatrix} \tag{2.22}$$

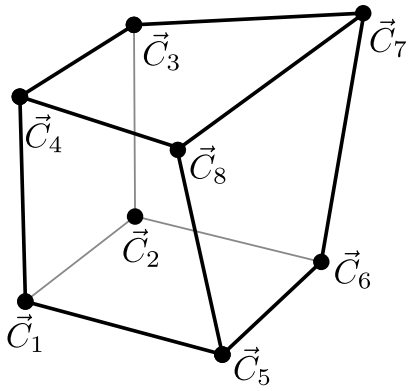


Figure 2.10: Vectors of coefficients for hexahedron element - stored nodes

The coefficient matrix is defined in a similar way for other basic element types. The equations for matrix coefficients can be obtained via symbolic integration, where e.g. a matlab symbolic toolbox[20] or a SymPy[21] package of Python can be used. A detailed description of this procedure is presented in Appendix A.4.

The matrix coefficients can be also derived by hand, although this method would lead to tedious and time-consuming calculations, and would be more prone to human error.

The discussed methodology allows accurate gradients at each primary mesh cell to be obtained. However, for the vertex-centred solver, the gradient field has to be 'transferred' from cells to nodes. This operation can be performed using volume weighting as presented in Eq. 2.23.

$$\nabla\phi_i = \sum_{i_c \in N_c(i)} (\nabla\phi_{i_c}\Omega_{i_c}) \Big/ \sum_{i_c \in N_c(i)} \Omega_{i_c} \tag{2.23}$$

$N_c(i)$ is a list of cells i_c coinciding at given node i. After replacing the cell gradient in Eq. 2.23 with Eq. 2.21, the calculations can be simplified as the cell volume (Ω_{i_c}) cancels out and only the nodal sum of element volumes - $\Omega_i^{sum} = \sum_{i_c \in N_c(i)} \Omega_{i_c}$ has to be stored. The final equation has the form Eq. 2.24.

$$\nabla\phi_i = \sum_{i_c \in N_c(i)} \left(\sum_{i \in N_n(i_c)} \left(\phi_i \vec{C}_i \right) \right) \Big/ \Omega_i^{sum} \tag{2.24}$$

[20] http://uk.mathworks.com/products/symbolic/?requestedDomain=www.mathworks.co
[21] http://www.sympy.org/en/index.html

2.3.6 Limiter

STAMPS uses slope limiters Ψ to prevent the generation of oscillations and spurious solutions in the regions of the computational domain with large gradients (e.g. at shocks). The limiting is applied during solution reconstruction step discussed in Section 2.3.2 (Eq. 2.14) by multiplying gradient vector by a scalar quantity Ψ. Limiter takes values between 0 and 1, where 0 indicates full limiting and the solution reconstruction becomes 1^{st}-order accurate, whereas value of 1 means no limiting is applied and the 2^{nd}-order accuracy is maintained.

The standard limiting approach proposed by Barth and Jespersen [121] was initially implemented in STAMPS - Eq. 2.25.

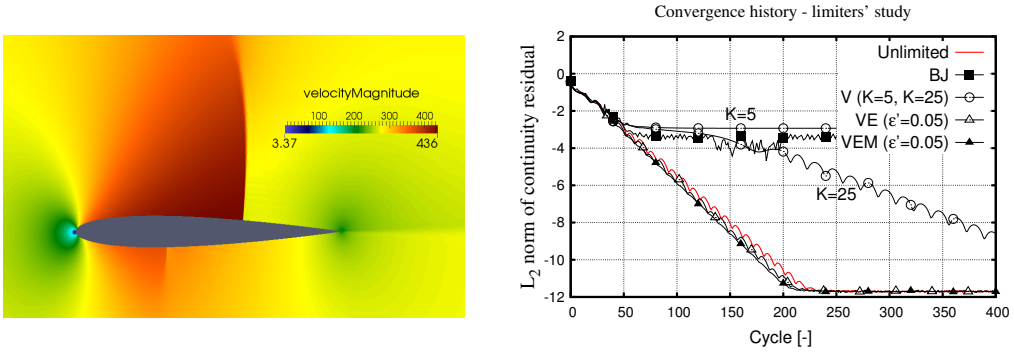
$$\widehat{\Psi}_i = \mathtt{min}_j \begin{cases} \mathtt{min}\left(1, \frac{\phi_{max}-\phi_i}{\Delta_i}\right), & if\ \Delta_i > w \\ \mathtt{min}\left(1, \frac{\phi_{min}-\phi_i}{\Delta_i}\right), & if\ \Delta_i < -w \\ 1.0, & if\ |\Delta_i| \leq w \end{cases} \tag{2.25}$$

$$\begin{aligned} \Delta_i &= 0.5\left(\nabla\phi_i \cdot \vec{r}_{ij}\right) \\ \phi_{max} &= \mathtt{max}\left(\phi_i, \mathtt{max}_j(\phi_j)\right) \\ \phi_{min} &= \mathtt{min}\left(\phi_i, \mathtt{min}_j(\phi_j)\right) \end{aligned} \tag{2.26}$$

The edge vector $\vec{r}_{ij} = \vec{X}(j) - \vec{X}(i)$ is defined as the difference between coordinates of node j and i; the $\mathtt{min}_j$ and $\mathtt{max}_j$ operators stand for the minimum and maximum value among the immediate neighbours j of node i; ϕ is a scalar field (e.g. density, x-velocity, etc.); $w = 10^{-12}$ is a constant used to prevent division by zero.

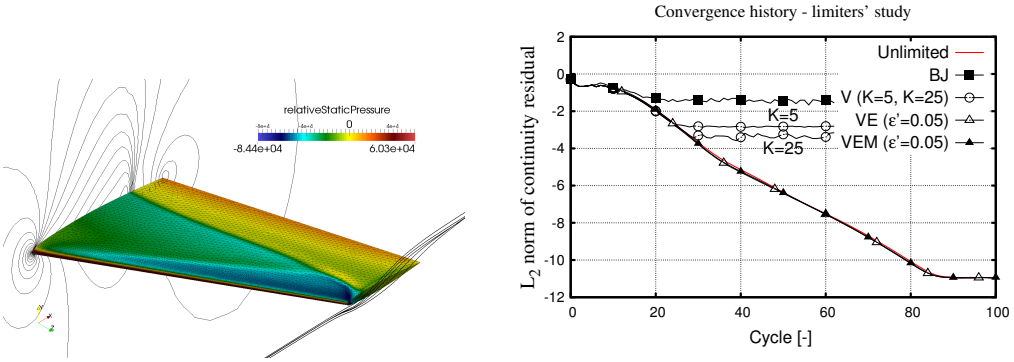
Although the Barth and Jespersen (BJ) limiter is simple, it introduces several discontinuities (non-differentiabilities) associated with the minimum and maximum functions which often lead to problems with convergence [51, 102]. A simple Naca aerofoil example at transonic conditions (Figure 2.12a) is used to show the convergence stall of residual norm around the level of -3 for the Barth and Jespersen limiter (Figure 2.11a). Similar behaviour is also visible on Figure 2.12b

where the 3D transonic Onera M6 wing case was used (Ma = 0.8395, $AoA = 3.06^{\circ}$ - Figure 2.12a).



(a) Velocity contour (VEM limiter) (b) Convergence comparison

Figure 2.11: Transonic Naca - convergence comparison for various limiters



(a) Pressure contour (VEM limiter) (b) Convergence comparison

Figure 2.12: Transonic M6 - convergence comparison for various limiters

The main discontinuity that leads to the convergence problem is associated with the definition of the limiter function itself [54], i.e. $\texttt{min}\left(1, \frac{\phi_{max}-\phi_i}{\Delta_i}\right)$ and $\texttt{min}\left(1, \frac{\phi_{min}-\phi_i}{\Delta_i}\right)$ from Eq. 2.25. A further problem associated with the BJ limiter is its activation due to numerical noise in smooth flow regions.

Venkatakrishnan [51] proposed a smooth alternative to the BJ limiter that replaces the minimum functions $\texttt{min}\left(1, \frac{\phi_{max}-\phi_i}{\Delta_i}\right)$ and $\texttt{min}\left(1, \frac{\phi_{min}-\phi_i}{\Delta_i}\right)$ from Eq. 2.25 with continuous alternatives shown in Eqs. 2.27 and 2.28.

$$\frac{f_{max}^2 + 2f_{max}}{f_{max}^2 + f_{max} + 2}, \text{ where } f_{max} = \frac{\phi_{max} - \phi_i}{\Delta_i} \tag{2.27}$$

$$\frac{f_{min}^2 + 2f_{min}}{f_{min}^2 + f_{min} + 2}, \text{ where } f_{min} = \frac{\phi_{min} - \phi_i}{\Delta_i} \tag{2.28}$$

The complete formula for the Venkatakrishnan limiter denoted (V) is presented in Eq. 2.29.

$$\widehat{\Psi}_i = \mathtt{min}_j \begin{cases} \frac{1}{\Delta_i}\frac{(\Delta^2_{j,max}+\epsilon^2)\Delta_i+2\Delta^2_i\Delta_{j,max}}{\Delta^2_{j,max}+2\Delta^2_i+\Delta_{-j}\Delta_{j,max}+\epsilon^2}, & if\ \Delta_i > w \\ \frac{1}{\Delta_i}\frac{(\Delta^2_{j,min}+\epsilon^2)\Delta_i+2\Delta^2_i\Delta_{j,min}}{\Delta^2_{j,min}+2\Delta^2_i+\Delta_i\Delta_{j,min}+\epsilon^2}, & if\ \Delta_i < -w \\ 1.0, & if\ |\Delta_i| \leq w \end{cases} \tag{2.29}$$

The Δ variables from Eq. 2.29 are defined in Eq. 2.30, where the indices i and j are nodes forming edge ij, and variables $\phi_{i,max}$ and $\phi_{i,min}$ are the local extremes among the immediate neighbours j of node i. Similarly $\phi_{j,max}$ and $\phi_{j,min}$. All the minimum and maximum values ($\phi_{i,max}$, $\phi_{i,min}$, $\phi_{j,max}$, and $\phi_{j,min}$) are pre-computed before calculating limiter Ψ.

$$\begin{aligned} \Delta_i &= 0.5\nabla\phi_i \cdot \vec{r}_{ij} \\ \Delta_{j,max} &= \phi_{j,max} - \phi_j \\ \Delta_{i,max} &= \phi_{i,max} - \phi_i \\ \Delta_{j,min} &= \phi_{j,min} - \phi_j \\ \Delta_{i,min} &= \phi_{i,min} - \phi_i \end{aligned} \tag{2.30}$$

The additional variable ϵ^2 that is present in Eq. 2.29 is defined in Eq. 2.31. This modification is recommended by Venkatakrishnan [52] to avoid activation of limiter in regions of nearly uniform flow, which could lead to an unnecessary reduction is spatial discretisation accuracy. Ω in Eq. 2.31 is a control volume and K is a user defined constant. In smooth regions of flow, variables Δ_{max} and Δ_{min} become of similar magnitude as characteristic cell size $\Omega^{1/3}$. Hence, in the near constant flow regions ϵ^2 will dominate terms Δ^2_{max} and Δ^2_{min} from Eq. 2.29 and the limiter reduces to 1 (no limiting). Constant K is used to control the strength of limiting where $K = 0$ corresponds to the fully limited solution, and $K = +\infty$ to the unlimited solution. Usually $K \in (5, 100)$ is used.

$$\epsilon^2 = K^3\Omega \tag{2.31}$$

Even though the V-limiter leads to an improved convergence as compared to the BJ limiter - Figures 2.11b and 2.12b - it still requires larger number of cycles to converge versus the unlimited case, and is sensitive to the choice of the constant K. In particular, for meshes where the neighbouring cell sizes differ significantly e.g. by factor 2, the ϵ^2 will differ by factor 8 which can lead to a convergence stall of residual norm. Wang proposed a further adjustment that resolves this issue [53]. It redefines the variable ϵ from Eq. 2.29 to be dependent on the global maximum and minimum of field ϕ and a constant $\epsilon' \in (0.01, 0.2)$ - Eq. 2.32. By default $\epsilon' = 0.05$ is used in STAMPS, where $\epsilon' = +\infty$ corresponds to the unlimited case. A short notation (VE) is used for this limiter.

$$\epsilon = \epsilon'(\phi_{global-max} - \phi_{global-min}) \tag{2.32}$$

It is evident from Figures 2.11b and 2.12b that the enhancement proposed by Wang (the VE limiter) leads to good convergence with an almost identical rate as compared to the unlimited case. For the NACA example, the required number of cycles is even slightly reduced when using the VE limiter, compared to the unlimited case. This can be explained by the limiter acting also as a local switch between 2^{nd}-order and 1^{st}-order solution accuracy, where the 1^{st}-order usually leads to a less stiff system of equations and is thus easier to converge. As the limiter decreases to zero a 1^{st}-order discretisation accuracy is locally obtained. A solution error convergence study would be required to investigate influence of limiter on the solution accuracy. However, this is outside the scope of this research and is therefore not discussed further.

Even though the VE limiter seems to be very robust based on the presented examples, it was reported by STAMPS users and the author that for some cases it still resulted in convergence stall. The additional adjustment was hence introduced by Michalak [54], where a smooth σ-function (Eq. 2.34) and a polynomial of degree 3 shown in Eq. 2.36 are proposed. The short notation (VEM) is used for this limiter. Note that Michalak used $\epsilon = \sqrt{K^3\Omega_i}$ while this research is using the modification proposed by Wang - Eq. 2.32. Variable $\Delta\phi_i$ is the difference

between local maximum and local minimum value of field ϕ at node i.

$$\Psi_i = \sigma_i + (1 - \sigma_i)\widehat{\Psi}_i \tag{2.33}$$

$$\sigma_i = \begin{cases} 1.0, & if \ \ (\Delta\phi_i \leq \epsilon) \\ s(y), & if \ \ (\epsilon > \Delta\phi_i > 2\epsilon) \\ 0.0, & if \ \ (\Delta\phi_i \geq 2\epsilon) \end{cases} \tag{2.34}$$

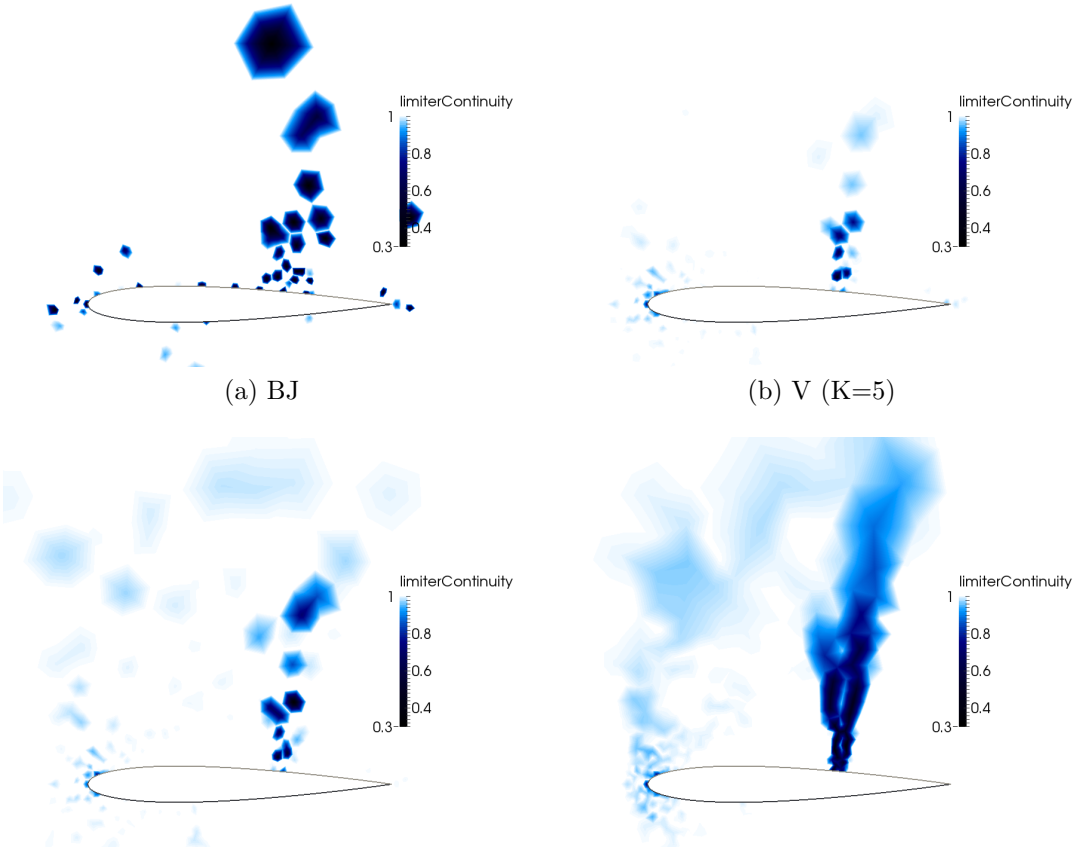
$$s(y) = 2y^3 - 3y^2 + 1 \tag{2.35}$$

$$y = \frac{\Delta\phi_i^2 - \epsilon^2}{\epsilon^2} \tag{2.36}$$

$$\Delta\phi_i = \phi_{i,max} - \phi_{i,min} \tag{2.37}$$

The VEM limiter is used as a default option in STAMPS, and it was proved to be most robust among other options, even though for the presented examples (Figure 2.11b and 2.12b) the VE and VEM limiters give almost identical convergence rate.

Figure 2.13 presents continuity limiter plots for the transonic NACA aerofoil example. The regions where the limiter is active are coloured blue. When the limiter value is equal to 0 the discretisation scheme changes to 1^{st}-order accurate, when it is 1 the standard 2^{nd}-order discretisation is used. From the plots one can see that the VEM limiter field (Figure 2.13d) is more smoothly distributed within the computational domain as compared to other limiters, but much more globally active. Hence some reduction in accuracy is expected.



(a) BJ (b) V (K=5)

(c) VE ($\epsilon' = 0.05$) (d) VEM ($\epsilon' = 0.05$)

Figure 2.13: Contour plots of the limiter fields for continuity equation (blue colour shows regions where the limiter is active)

Apart from the treatment of shocks for transonic and supersonic flows, the VEM limiter also helps to stabilise convergence by clipping high gradients that often arise at the intermediate state of a converging flow. For this reason, the VEM limiter is useful for use with any flow conditions, including subsonic cases. Figure 2.14 shows the density limiter for the subsonic Rae2822 case. Excluding the sharp trailing edge regions where a strong limiting is observed ($\Psi \approx 0.11$) and wake behind the aerofoil ($\Psi \approx 0.7 - 0.9$) the limiter is equal to 1. There are no visible differences when comparing flow solution (contour plots) between the unlimited case and the VEM case; however, a solution error convergence study would be required to investigate the impact of the VEM limiter on the solution accuracy.

Eq. 2.14 in Section 2.3.2 shows that the limiter acts as a switch between 2^{nd}-order and 1^{st}-order spatial discretisation by multiplying gradients by a constant Ψ

that can have values between 0 (1^{st}-order reconstruction) and 1 (2^{nd}-order reconstruction). A reduced order of accuracy of solution reconstruction can affect the overall accuracy of the spatial discretisation scheme. An example investigation of the influence of slope limiters on the accuracy was carried out by Hubbard [89]. Michalak [54] also briefly discusses the influence of the limiter similar to the VEM (default in STAMPS) on the accuracy of the discretisation scheme suggesting its low dissipation; however, no evidence is provided to show the preserved nominal discretisation accuracy of the original scheme.

The influence of the limiters used in STAMPS on the accuracy of the spatial discretisation scheme is outside the scope of this work and will not be discussed further.

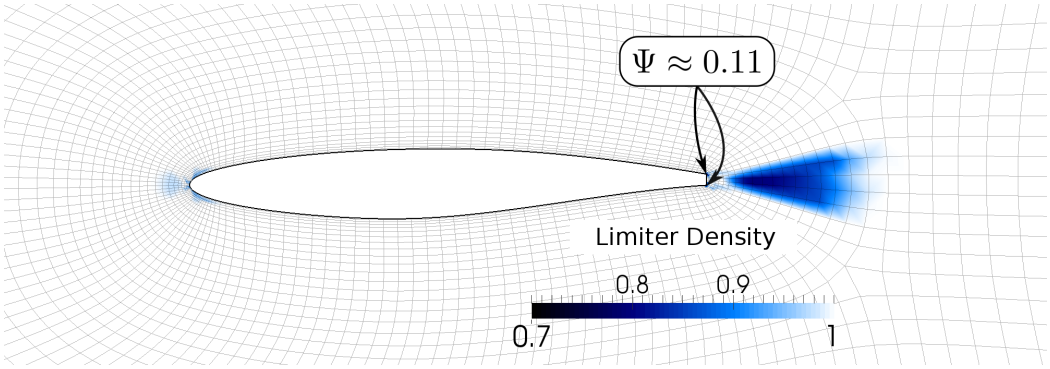


Figure 2.14: Subsonic Rae2822 (Ma = 0.3, $AoA = 0^{\circ}$) - contour plots of the VEM limiter for density (blue colour shows regions where the limiter is active)

2.3.7 Convective flux

Having all geometric quantities defined (Section 2.3.1) and solution reconstruction on the flux face introduced (Section 2.3.2) a numerical flux integration can be performed. First the convective flux integration is considered.

The total convective flux is a sum of fluxes through the dual-cell-faces (Figure 2.5) denoted $\vec{F}_{c,ij}$, which are often called edge fluxes to reflect the fact that the edge-based data structure is employed for the discretisation. The appropriate boundary contribution F_{c,i_b} has to be added for the boundary volumes.

$$\vec{F}_{c,i} = \sum_{j \in N_e(i)} \vec{F}_{c,ij} + \begin{cases} 0, & N_b(i) = \varnothing \\ \sum_{i_b \in N_b(i)} \vec{F}_{c,i_b}, & N_b(i) \neq \varnothing \end{cases} \tag{2.38}$$

In STAMPS's, the ROE flux-difference splitting scheme [36] is used by default, and will be described in this Section. For a detailed information on AUSM$^{up}_{+}$ flux implementation refer to [50].

The ROE flux is evaluated using a central difference scheme with added matrix-valued artificial viscosity - Eq. 2.39, where $ds_{ij} = |\vec{s}_{ij}|$, and L and R are the left and right state (Figure 2.7), respectively.

$$\vec{F}_{c,ij} = \frac{1}{2}\left[\vec{F}_{c,R} + \vec{F}_{c,L} - |A_{ROE}|\left(\vec{W}_R - \vec{W}_L\right) ds_{ij}\right] \tag{2.39}$$

The left and right convective fluxes ($\vec{F}_{c,L}$, $\vec{F}_{c,R}$) are evaluated using Eq. 2.40 with $\vec{f}_c$ defined in Eq. 2.3. The left $\vec{U}_L$ and right $\vec{U}_R$ comes from the solution reconstructed as described in Section 2.3.2.

$$\vec{F}_{c,R} = \vec{f}_c\left(\vec{U}_R\right) ds_{ij}, \quad \vec{F}_{c,L} = \vec{f}_c\left(\vec{U}_L\right) ds_{ij} \tag{2.40}$$

The last term in Eq. 2.39, is based on the decomposition of flux difference over a face of control volume into the sum of wave contributions while ensuring conservation property. The so called ROE matrix A_{ROE} is introduced, which is identical to the convective flux Jacobian (see e.g. A.9 in [102]). The product of the ROE matrix with the jump in conservative variables $\vec{W}$ can be efficiently calculated using Eq. 2.41, where all components are defined in Eqs. 2.42-2.45. In the current implementation in STAMPS, the SA equation is decoupled from other equations i.e. the 6^{th} column and 6^{th} row of matrix A_{ROE} is set to zero except for the diagonal term. See e.g. [56] for the ROE matrix coupling with the SA model. The ROE matrix is evaluated for the so-called ROE averaged variables $\widetilde{\vec{U}} = [\widetilde{\rho}, \widetilde{u}, \widetilde{v}, \widetilde{w}, \widetilde{h}_t, \widetilde{\nu}]^T$ defined as a function of left and right state - see Appendix A.3. The expression $\Delta(\bullet) = (\bullet)_R - (\bullet)_L$ defines the jump condition, where the $(\bullet)$ is to be replaced with each variable from flow state vector $\vec{U}$.

$$|A_{ROE}|\left(\vec{W}_R - \vec{W}_L\right) = \Delta\vec{f}_1 + \Delta\vec{f}_{2,3,4} + \Delta\vec{f}_5 + \Delta\vec{f}_6 \tag{2.41}$$

$$\Delta\vec{f_1} = \left|\widetilde{V} - \widetilde{c}\right| \left(\frac{\Delta p - \widetilde{\rho}\widetilde{c}\Delta V}{2\widetilde{c}^2}\right) \begin{bmatrix} 1 \\ \widetilde{u} - \widetilde{c}n_x \\ \widetilde{v} - \widetilde{c}n_y \\ \widetilde{w} - \widetilde{c}n_z \\ \widetilde{h_t} - \widetilde{c}\widetilde{V} \\ 0 \end{bmatrix} \tag{2.42}$$

$$\Delta\vec{f}_{2,3,4} = \left|\widetilde{V}\right| \left(\Delta\rho - \frac{\Delta p}{\widetilde{c}^2}\right) \left\{ \begin{bmatrix} 1 \\ \widetilde{u} \\ \widetilde{v} \\ \widetilde{w} \\ \widetilde{q}^2/2 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ \Delta u - \Delta V n_x \\ \Delta v - \Delta V n_y \\ \Delta w - \Delta V n_z \\ \widetilde{\vec{U}_v} \cdot \Delta\vec{U}_v - \widetilde{V}\Delta V \\ 0 \end{bmatrix} \right\} \tag{2.43}$$

$$\Delta\vec{f_5} = \left|\widetilde{V} + \widetilde{c}\right| \left(\frac{\Delta p + \widetilde{\rho}\widetilde{c}\Delta V}{2\widetilde{c}^2}\right) \begin{bmatrix} 1 \\ \widetilde{u} + \widetilde{c}n_x \\ \widetilde{v} + \widetilde{c}n_y \\ \widetilde{w} + \widetilde{c}n_z \\ \widetilde{h_t} + \widetilde{c}\widetilde{V} \\ 0 \end{bmatrix} \tag{2.44}$$

$$\Delta\vec{f_6} = \left|\widetilde{V}\right| \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ \Delta\widehat{\nu} \end{bmatrix} \tag{2.45}$$

The first terms present in Eqs. 2.42 - 2.44 i.e. $|\widetilde{V} - \widetilde{c}|$, $|\widetilde{V}|$, $|\widetilde{V} + \widetilde{c}|$ are the eigenvalues of matrix A_{ROE}, denoted Λ_{ROE}. The eigenvalues Λ_{ROE} are modified

using Harten's entropy correction [57] as presented in Eq. 2.46. The correction is required to rule out non-physical solutions such as carbuncles that can arise with ROE flux scheme when applied to hyperbolic equations, e.g. supersonic flows. The carbuncle phenomena was first reported by Peery and Imlay [119] on a hypersonic flow past the cylinder. Instead of a smooth bow shock in front of the cylinder a solution with two oblique shocks was observed upstream the stagnation region - a carbuncle. A more recent discussion on this phenomenon can be found in [58]. At the time of writing only a simple Harten entropy correction is implemented in STAMPS which is an effective yet dissipative technique. Several other methods were proposed to tackle the carbuncle problem [59, 60, 61], and more recently the entropy-stable flux proposed by Ismail et. al. [115].

$$|\Lambda_{ROE}| = \begin{cases} |\Lambda_{ROE}|, & if\ |\Lambda_{ROE}| > \delta \\ \frac{\Lambda_{ROE}^2+\delta^2}{2\delta}, & if\ |\Lambda_{ROE}| \leq \delta \end{cases} \tag{2.46}$$

The variable δ in Eq. 2.46 is usually set to some fraction of the local speed of sound ($\widetilde{c}$). In STAMPS, the value of $0.05\widetilde{c}$ is used by default. The treatment presented in Eq. 2.46 is applied to all three eigenvalues including eigenvalue $|\widetilde{V}|$ in order to prevent linear waves $\Delta\vec{f}_{2,3,4}$ (see Eq. 2.43) from disappearing as $\widetilde{V} \to 0$.

A boundary contribution has to be added for the boundary control volumes (see Eq. 2.38). This contribution is calculated differently depending on the boundary patch type.

- Walls (adiabatic). As a result of zero normal velocity $V_{i_b} = \vec{U}_{v,i_b} \cdot \vec{n_b} = 0$ (for viscous walls the velocity vector $\vec{U}_{v,i_b}$ is zero itself) there is no boundary flux except for the pressure-related terms in momentum equations (see Eq. 2.47). The global node ID i required to access pressure at the boundary node i_b, is obtained using the connectivity $bNode2node(i_b)$, $\vec{n}_b = \vec{s}_{i_b}/ds_{i_b}$ is wall

normal, and $ds_{i_b} = |\vec{s}_{i_b}|$ is wall face area.

$$\vec{F}_{c,i_b} = \begin{bmatrix} 0 \\ n_{b,x}\, p_{i_b} \\ n_{b,y}\, p_{i_b} \\ n_{b,z}\, p_{i_b} \\ 0 \\ 0 \end{bmatrix} ds_{i_b} \tag{2.47}$$

- Inlet/outlet, freestream. The right state $\vec{U}_R$ is set to the value at the ghost node associated with given boundary node i_b. The flux is then evaluated as is done for the interior flux face (see Eq. 2.39). The ghost state is set as described in Section 2.3.10.

- Periodic and symmetry: no flux contributions.

2.3.8 Viscous flux

An edge-based numerical integration is also performed for viscous flux integration, with an additional boundary contribution for the boundary control volumes - Eq. 2.48.

$$\vec{F}_{v,i} = \sum_{j \in N_e(i)} \vec{F}_{v,ij} + \begin{cases} 0, & N_b(i) = \varnothing \\ \sum_{i_b \in N_b(i)} \vec{F}_{v,i_b}, & N_b(i) \neq \varnothing \end{cases} \tag{2.48}$$

The viscous flux for the internal faces is evaluated using Eq. 2.49 with the flow state vector $\vec{U}_{ij}$ and gradient of the flow variables reconstructed at the flux face.

$$\vec{F}_{v,ij} = \vec{f_v}\left(\vec{U}_{ij}, \nabla\vec{U}_{ij}\right) ds_{ij} \tag{2.49}$$

The viscous flux function $\vec{f_v}$ is defined in Eq. 2.4.

The flux face state is reconstructed using simple average of nodal values i and j - Eq. 2.50.

$$\vec{U}_{ij} = 0.5\left(\vec{U}_L + \vec{U}_R\right), \quad \vec{U}_L = \vec{U}_i, \quad \vec{U}_R = \vec{U}_j \tag{2.50}$$

Reconstruction of the gradients requires more attention as the simple average obtained as shown in Eq. 2.51 leads to decoupling of the solution for hexahedral grids and a wide stencil with unfavourable distribution of weights [4]. Gradients at nodes i and j are computed using a Green-Gauss approach as described in Section 2.3.3.

These decoupling problems can be fixed by using directional derivative along the edge - see Eq. 2.52 - and obtaining the final reconstructed gradient as presented in Eq. 2.53. Vector $\vec{t}_{ij}$ is an edge tangent direction and l_{ij} edge length. The notation $\nabla\vec{U}_{ij}$ denotes the gradient of flow variables for each variable from vector $\vec{U}$. Additionally, the gradient of the temperature field has to be computed for the thermal conductivity term present in RANS equations.

$$\nabla\vec{U}_{ij}^{avg} = 0.5\left(\nabla\vec{U}_i + \nabla\vec{U}_j\right) \tag{2.51}$$

$$\left(\frac{\partial U}{\partial l}_{ij}\right) = \frac{\vec{U}_j - \vec{U}_i}{l_{ij}} \tag{2.52}$$

$$\nabla\vec{U}_{ij} = \nabla\vec{U}_{ij}^{avg} - \left[\nabla\vec{U}_{ij}^{avg}\cdot\vec{t}_{ij} - \left(\frac{\partial U}{\partial l}_{ij}\right)\right]\vec{t}_{ij} \tag{2.53}$$

$$\vec{t}_{ij} = \frac{\vec{r}_{ij}}{l_{ij}}, \quad \vec{r}_{ij} = \vec{X}_\text{J} - \vec{X}_i \tag{2.54}$$

As with convective flux, the boundary contribution for the viscous flux has to be added for the boundary control volumes - Eq. 2.48. This contribution is different for each boundary patch type

- walls (adiabatic): no viscous flux is calculated for the strong/hard boundary condition. The momentum and turbulence residual for the hard wall boundary condition is set to zero after the flux accumulation is completed.
- inlet/outlet, freestream. The flux is evaluated as is done for the interior flux face - Eq. 2.49, with the state vector $\vec{U}_j$ in Eq. 2.50 replaced with the state at the ghost node associated with the given boundary node i_b. The ghost state is set as described in the boundary condition Section 2.3.10. No reconstruction is required for the gradient as the value at node i is used

directly i.e. $\nabla\vec{U}_{ij} = \nabla\vec{U}_i$.

- periodic and symmetry: no flux contributions.

2.3.9 Volume sources

The integral of volume source from Eq. 2.1 can be evaluated for node i using the nodal values of the source term vector $\vec{q}_i$) and the volume of the dual cell Ω_i - Eq. 2.55. In the current implementation in STAMPS, there is a non-zero volume source for the Spalart-Allmaras equation as presented in Appendix A.2. and for the method of manufactured solution as shown in Section A.5.

$$\int_{\Omega_i} \vec{q}\, d\Omega_i \approx \vec{Q}_i = \vec{q}_i\, \Omega_i \tag{2.55}$$

2.3.10 Boundary conditions (BCs)

The treatment of boundary fluxes was introduced in Section 2.3.7 and 2.3.8; however, some boundary types (far-field, inlet, outlet) require setting the boundary state vector $\vec{U}_{i_b}$, which is discussed in detail in this section.

The expression '*ghost state*' is often used for the vector $\vec{U}_{i_b}$. This state is set before the flux accumulation takes place. Figure 2.15 presents the boundary dual volume constructed for a simple 2D example.

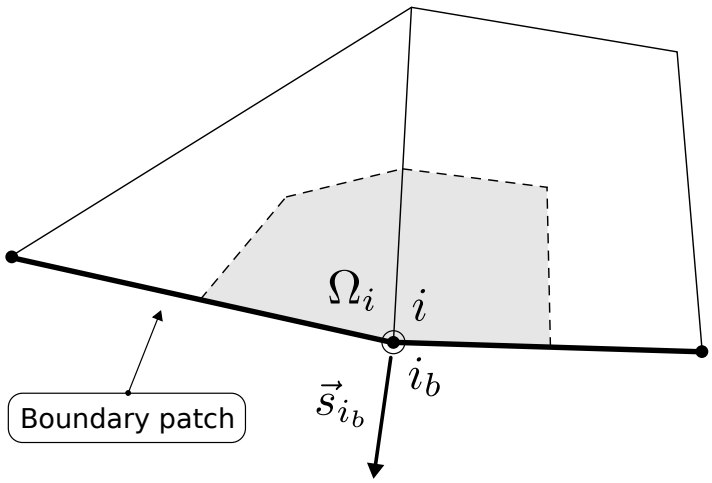


Figure 2.15: Boundary definition - 2D example. Ω_i - dual cell, i_b - boundary node ID, i - mesh/global node ID i.e. $i = bNode2node(i_b)$, $\vec{s}_{i_b}$ - boundary weight

In STAMPS, the far-field, inlet, and outlet boundary types are imposed weakly, this is, by calculating the boundary flux contribution to the residual rather than setting the state vector explicitly. The procedure for boundary flux calculation for far-field, inlet and outlet boundary conditions (see Eqs. 2.56-2.57) is as follows:

- Obtain the ghost state $\vec{U}_{i_b}$ for each boundary condition type. In general, the ghost state is a function of free-stream state $\vec{U}_\infty$, interior state $\vec{U}_i$, and set of user inputs denoted $\mathcal{B}$ which are dependent on the BC type (e.g. flow direction unit vector $\vec{n}_\infty$).

$$\vec{U}_{i_b} = f\left(\vec{U}_i, \vec{U}_\infty, \mathcal{B}\right) \tag{2.56}$$

- Set left and right state vector for flux calculation

$$\begin{aligned} \vec{U}_L &\leftarrow \vec{U}_i \\ \vec{U}_R &\leftarrow \vec{U}_{i_b} \end{aligned} \tag{2.57}$$

- Calculate fluxes as it is done for the interior control volumes - see Sections 2.3.7 and 2.3.8.

The following subsections discuss details of implementation of boundary conditions available in STAMPS.

Far-field

The far-field boundary condition is specified using a set of user inputs:

- Mach number: M_∞ $[-]$
- Flow direction normal: $\vec{n}_\infty$ $[-]$
- Far-field pressure: p_∞ $[Pa]$
- Far-field absolute temperature: T_∞ $[K]$

Based on the provided information, the complete free-stream state vector is computed as presented in Eq. 2.58, where the speed of sound is calculated using ideal

gas assumption ($c_\infty = \sqrt{\gamma R T_\infty}$), and the laminar dynamic viscosity ($\mu_{L,\infty}$) using Southerland's law - Appendix A.2, Eq. A.22. For the definition of the remaining constants see Appendix A.1. The variables in column $\vec{U}_i$ of Table 2.1 stand for the current solution at the node i.

$$\vec{U}_\infty = \begin{bmatrix} \rho_\infty \\ u_\infty \\ v_\infty \\ w_\infty \\ p_\infty \\ \widehat{\nu}_\infty \end{bmatrix} \tag{2.58}$$

Finally, the complete treatment of the far-field boundary condition is summarised in Table 2.1. It shows how the left/right (i/i_b) states are set and what boundary flux contributions are added to the residual at node $i = bNode2node(i_b)$. Column $\vec{R}_i^{post}$ shows the additional treatment of the residual after all fluxes are accumulated - used whenever applicable. The hyphen (-) indicates that either no treatment is required for the given quantity or that the quantity is not needed for the given boundary condition type.

Equation / Variable	$\vec{U}_i$	$\vec{U}_{i_b}$	$\vec{F}_{i_b}$	$\vec{R}_i^{post}$
ρ	ρ_i^n	ρ_∞	$\mathbf{f}_i$	-
u	u_i^n	u_∞	$\mathbf{f}_i$	-
v	v_i^n	v_∞	$\mathbf{f}_i$	-
w	w_i^n	w_∞	$\mathbf{f}_i$	-
p	p_i^n	p_∞	$\mathbf{f}_i$	-
$\widehat{\nu}$	$\widehat{\nu}_i^n$	$\widehat{\nu}_\infty$	$\mathbf{f}_i$	-

Table 2.1: Far-field boundary condition treatment. Notation $\mathbf{f}_i$ indicates that the boundary flux is calculated the same way as the interior flux (see Sections 2.3.7 and 2.3.8)

Subsonic inlet/outlet

For both subsonic inlet and outlet, the states at node i (left) are defined using the current solution (at the pseudo-time n). The ghost state (right) is set based

on the incoming and outgoing characteristics, and is different for subsonic inlet (Tables 2.2 - 2.3).

For the subsonic inlet, all eigenvalues of RANS system of equations are positive except for eigenvalue one. The positive eigenvalue corresponds to waves entering the domain and the negative eigenvalue to waves leaving the domain, which means that all variables at the subsonic inlet boundary have to be supplied and one variable is taken from the interior. The subsonic inlet boundary condition is summarised in Table 2.2 [22].

Equation / Variable	$\vec{U}_i$	$\vec{U}_{i_b}$	$\vec{F}_{i_b}$	$\vec{R}_i^{post}$
ρ	ρ_i^n	$f\left(\rho_\infty, \rho_i^n, \mathcal{B}\right)$	$\mathbf{f}_i$	-
u	u_i^n	$f\left(u_\infty, u_i^n, \mathcal{B}\right)$	$\mathbf{f}_i$	-
v	v_i^n	$f\left(v_\infty, v_i^n, \mathcal{B}\right)$	$\mathbf{f}_i$	-
w	w_i^n	$f\left(w_\infty, w_i^n, \mathcal{B}\right)$	$\mathbf{f}_i$	-
p	p_i^n	$f\left(p_\infty, p_i^n, \mathcal{B}\right)$	$\mathbf{f}_i$	-
$\widehat{\nu}$	$\widehat{\nu}_i^n$	$f\left(\widehat{\nu}_\infty, \widehat{\nu}_i^n, \mathcal{B}\right)$	$\mathbf{f}_i$	-

Table 2.2: Subsonic inlet boundary condition. Notation $\mathbf{f}_i$ indicates that the boundary flux is obtained same as for the interior (see Sections 2.3.7—2.3.8)

For the subsonic outlet all eigenvalues of the RANS system of equations are negative except for eigenvalue $(V + c)$. This means that all variables at the subsonic outlet boundary have to be taken from the interior (current solution) except for static pressure which is imposed $p_{i_b} = p_\infty$ - see Table 2.3, column $\vec{U}_{i_b}$. The freestream static pressure (p_∞) is provided by the user in the input file. The temperature T_i^n is calculated using ideal gas law i.e. $T_i^n = \gamma p_i^n / \rho_i^n$. The velocity vector is defined using Eq. 2.59 in order to prevent reverse flows.

$$\vec{U}_{v,i_b} = f\left(\vec{U}_{v,i}^n, \vec{n}_{i_b}\right) = \begin{cases} \vec{U}_{v,i}^n, & if\ V > 0 \\ \\ |V|\vec{n}_{i_b}, & if\ V \leq 0 \end{cases}, \quad V = \vec{U}_{v,i}^n \cdot \vec{n}_{i_b}, \quad \vec{n}_{i_b} = \frac{\vec{s}_{i_b}}{|\vec{s}_{i_b}|} \tag{2.59}$$

[22] Further details on the calculations of boundary state vector $\vec{U}_{i_b}$ can be found in [116].

Equation / Variable	$\vec{U}_i$	$\vec{U}_{i_b}$	$\vec{F}_{i_b}$	$\vec{R}_i^{post}$
ρ	ρ_i^n	$\gamma p_\infty / T_i^n$	$\mathbf{f}_i$	-
u	u_i^n	$f\left(\vec{U}_{v,i}^n, \vec{n}_{i_b}\right)$	$\mathbf{f}_i$	-
v	v_i^n	$f\left(\vec{U}_{v,i}^n, \vec{n}_{i_b}\right)$	$\mathbf{f}_i$	-
w	w_i^n	$f\left(\vec{U}_{v,i}^n, \vec{n}_{i_b}\right)$	$\mathbf{f}_i$	-
p	p_i^n	p_∞	$\mathbf{f}_i$	-
$\widehat{\nu}$	$\widehat{\nu}_i^n$	$\widehat{\nu}_i^n$	$\mathbf{f}_i$	-

Table 2.3: Subsonic outlet boundary condition treatment. Notation $\mathbf{f}_i$ indicates that the boundary flux is calculated the same way as the interior flux (see Sections 2.3.7 and 2.3.8)

Inviscid (slip) wall

There is no flow across the boundary surface for the inviscid walls, which formally can be written $\vec{U}_{v,i_b} \cdot \vec{n}_{i_b} = 0$, i.e. the velocity normal to the wall is zero. This also results in zero contravariant velocity V. Consequently, the flux vector reduces to pressure term alone and can be imposed weakly as previously presented in Eq. 2.47. The ghost state vector is not needed for this boundary type as the only required quantity is pressure, which is simply taken as a nodal value at the current time step i.e. $p_{i_b} \leftarrow p_i^n$. The treatment for an inviscid wall is summarised in Table 2.4.

Equation / Variable	$\vec{U}_i$	$\vec{U}_{i_b}$	$\vec{F}_{i_b}$	$\vec{R}_i^{post}$
ρ	ρ_i^n	-	-	-
u	u_i^n	-	$n_{b,x} p_{i_b} ds_{i_b}$	-
v	v_i^n	-	$n_{b,y} p_{i_b} ds_{i_b}$	-
w	w_i^n	-	$n_{b,z} p_{i_b} ds_{i_b}$	-
p	p_i^n	-	-	-
$\widehat{\nu}$	$\widehat{\nu}_i^n$	-	-	-

Table 2.4: Inviscid wall boundary condition treatment

Viscous (no-slip) wall

The relative velocity between the surface and the fluid is zero for this boundary type. In particular, for the zero wall velocity the Cartesian components are $u = v = w = 0$. This condition is imposed in a strong form at each solver iteration before the residual is calculated - see column $\vec{U}_i$ in Table 2.5. As a result, the momentum equation does not need to be solved at the boundary and hence no fluxes ($\vec{F}_{i_b}$) need to be calculated. Similar treatment is done for the turbulent equation where the SA variable $\widehat{\nu}$ is imposed to be zero. The ghost state vector is not required for this boundary type. The density and pressure are taken from the current approximation of the converging solution i.e. ρ_i^n, p_i^n, where n is the current pseudo time or iteration.

An adiabatic viscous wall is implemented in STAMPS, which means that the normal temperature gradient is zero which results in zero thermal conductivity term in energy equation (see Appendix A.2).

Note that the continuity, energy and SA equation fluxes are zero for the viscous wall boundary condition, whereas the non-zero momentum flux (pressure-related term) does not need to be calculated as the velocity vector is imposed in a strong manner.

Equation / Variable	$\vec{U}_i$	$\vec{U}_{i_b}$	$\vec{F}_{i_b}$	$\vec{R}_i^{post}$
ρ	ρ_i^n	-	-	-
u	$\mathbf{0}$	-	-	$\mathbf{0}$
v	$\mathbf{0}$	-	-	$\mathbf{0}$
w	$\mathbf{0}$	-	-	$\mathbf{0}$
p	p_i^n	-	-	-
$\widehat{\nu}$	$\mathbf{0}$	-	-	$\mathbf{0}$

Table 2.5: Adiabatic viscous wall treatment

The order in which the viscous wall boundary condition is treated is presented for completeness:

- Set $u = v = w = 0$ and $\widehat{\nu} = 0$ at the viscous wall.
- Accumulate interior fluxes. There is no need to calculate wall boundary fluxes as the momentum and SA equations are not solved. Similarly, conti-

nuity and energy fluxes are zero for an adiabatic wall as the contravariant velocity V and wall normal temperature gradient are zero.

- Set momentum and turbulence residuals to zero at the viscous wall to prevent generation of non-zero velocity and turbulent variables - see Table 2.5, column $\vec{R}_i^{post}$.
- Update density and pressure variables.
- Repeat until the solution has converged.

Symmetry

For the symmetry boundary condition, there is no boundary flux across the surface. The symmetry plane normal is assumed to be aligned with z-coordinate in STAMPS - therefore appropriate checks are implemented. The z-velocity component w is set to zero. Similarly, the residual for z-momentum equation is zeroed at the symmetry nodes after the flux accumulation is completed. The ghost state vector is not required for this boundary type.

It is necessary to correct edge weights of those faces of the control volume which touch the boundary. All components of boundary edge weights that are normal to the symmetry plane are zeroed - see Eq. 2.60. Furthermore, the gradients have to be corrected as presented in Eq. 2.61. The subscript in the equation indicates components of the gradient to be zeroed.

$$\vec{s}_{ij} \leftarrow \vec{s}_{ij} - (\vec{s}_{ij} \cdot \vec{n}_{i_b})\, \vec{n}_{i_b} \tag{2.60}$$

Equation / Variable	$\vec{U}_i$	$\vec{U}_{i_b}$	$\vec{F}_{i_b}$	$\vec{R}_i^{post}$
ρ	ρ_i^n	-	-	-
u	u_i^n	-	-	-
v	v_i^n	-	-	-
w	$\mathbf{0}$	-	-	$\mathbf{0}$
p	p_i^n	-	-	-
$\widehat{\nu}$	$\widehat{\nu}_i^n$	-	-	-

Table 2.6: Symmetry plane

$$\begin{bmatrix} \nabla\rho_{i,z}^{n} \\ \nabla u_{i,z}^{n} \\ \nabla v_{i,z}^{n} \\ \nabla w_{i,xy}^{n} \\ \nabla p_{i,z}^{n} \\ \nabla \widehat{\nu}_{i,z}^{n} \\ \nabla T_{i,z}^{n} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{2.61}$$

Although in many CFD applications it is justified to use symmetry boundary condition this decision should be made with caution. Even though the computational domain may be symmetric the solution to flow equations may not be symmetric and the accuracy of a simulation may be affected.

Periodic

In the current implementation, STAMPS supports only node-to-node periodicity; that is, the mesh at the periodic patches has to be topologically consistent. Furthermore, only one periodic pair is allowed. Translational periodicity can be defined in any direction, whereas rotational periodicity requires rotation axis to be aligned with x-axis - Figure 2.16. The inputs for periodic patches that have to be provided by the user are as follows

- Rotational periodicity requires an angle (α) between lower and upper periodic patch. Lower periodic patch is always the first in the patch list. The periodicity angle is used to create the rotation matrix Eq. 2.62. This is required for an appropriate treatment of quantities at periodic nodes as described later in this section. It is a 6×6 matrix where empty entries are zero.

$$\Gamma = \begin{bmatrix} 1 & & & & & \\ & 1 & & & & \\ & & \cos(\alpha) & -\sin(\alpha) & & \\ & & \sin(\alpha) & \cos(\alpha) & & \\ & & & & 1 & \\ & & & & & 1 \end{bmatrix} \tag{2.62}$$

- Translational periodicity requires a translation vector ($\vec{p}$) from lower to upper periodic patch. The rotation matrix Γ becomes a unity matrix for this case.

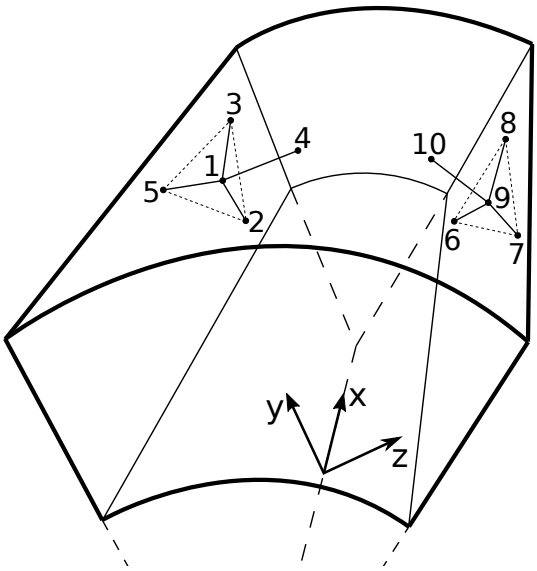


Figure 2.16: 3D rotational periodic case with periodic pair and its stencil

There are no boundary contributions to the residual for periodic BC and the volumes are treated as internal cells. An additional data structure obtained in a pre-processing step is required, as well as connectivity, an appropriate treatment of metrics, local time step, flow variables, gradients, residuals, and Jacobian. All necessary steps are described below.

First, a list of corresponding periodic nodes has to be created to allow for communication between lower and upper periodic boundaries. The connectivity is constructed in the pre-processing step based on the node coordinates. A pair of nodes (e.g. lower periodic node 1 and upper periodic node 9 in Figure 2.16) is accessed using a periodic edge loop (a virtual edge):

- $i = \texttt{periodicEdge2nodes}(1, i_{pe})$
- $j = \texttt{periodicEdge2nodes}(2, i_{pe})$.

Second, the dual volume from the lower periodic patch (Ω_i^l) and the dual volume from upper periodic patch (Ω_j^u) are first combined and then the obtained sum

is assigned to both nodes - see Eq. 2.63, and Figure 2.17. This step is necessary for appropriate treatment of the local time step and gradient (Section 2.3.3).

For the cell-based gradient, the sum of primal cell volumes (see Eq. 2.24) that coincide at the lower and upper periodic node have to be treated accordingly - Eq. 2.64

$$\Omega_i,\, \Omega_j \quad \leftarrow \quad \left(\Omega_i^l + \Omega_j^u\right) \tag{2.63}$$

$$\Omega_i^{sum},\, \Omega_j^{sum} \quad \leftarrow \quad \left(\Omega_i^{sum,l} + \Omega_j^{sum,u}\right) \tag{2.64}$$

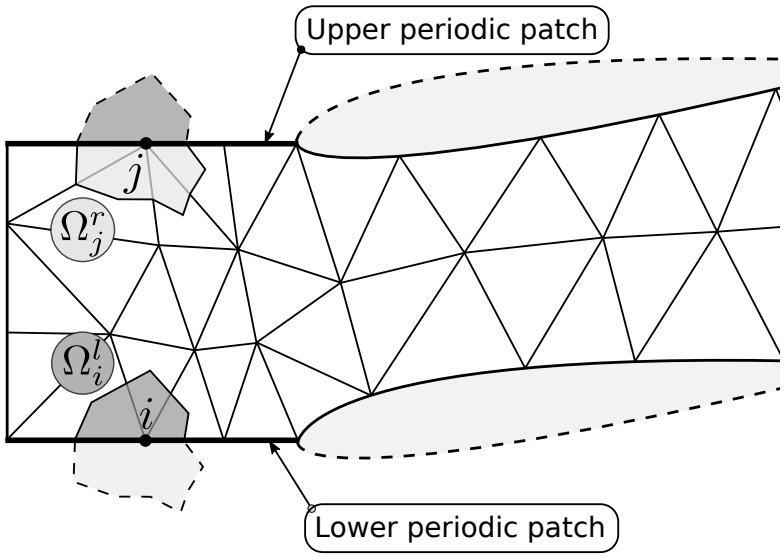


Figure 2.17: Blade section with lower and upper periodic volumes

Next, the local time step used for pseudo-time stepping requires attention for periodic nodes. The spectral radii Λ are first accumulated for lower and upper periodic nodes as shown in Eq. 2.65, where Λ_c is convective corresponds to the maximum eigenvalue of the convective flux Jacobian, Λ_v is maximum eigenvalue of the viscous flux Jacobian, and c_v is a constant set to 1 (refer Section 2.4). The boundary contributions are excluded because the periodic cells are treated as internal - see Figure 2.17.

$$\Lambda_i^l = (\Lambda_c + c_v\Lambda_v)_i^l \tag{2.65}$$

$$\Lambda_j^u = (\Lambda_c + c_v\Lambda_v)_j^u$$

The local time step for nodes i and j is calculated as shown in Eq. 2.66.

$$\Delta t_i,\, \Delta t_j \quad \leftarrow \quad \left(CFL\, \frac{\Omega_i^l + \Omega_j^u}{\Lambda_i^l + \Lambda_j^u} \right) \tag{2.66}$$

Next, to make sure that flow variables at the lower and upper periodic patches are consistent, the treatment presented in Eq. 2.67 and Eq. 2.68 has to be carried out. For the translational periodic, where the rotation matrix Γ is the identity matrix, the operations are equivalent to taking an average of flow variables at lower (i) and upper (j) periodic nodes. For the rotational periodicity the rotation matrix defined in Eq. 2.62 is used as the y-velocity and z-velocity components have to be rotated from upper to lower node before the summation takes place (Eq. 2.67), and then the obtain average is rotated back to the upper periodic node - Eq. 2.68. The X-velocity component remains unchanged due to the assumed alignment of the rotation axis with x-axis. These operations are required after the solution initialisation and after each pseudo-time step. The latter is only to prevent inconsistency between lower and upper periodic states due to the numerical precision in the solution update.

$$\vec{U}_i \quad \leftarrow \quad 0.5 \left(\vec{U}_i^l + \Gamma\, \vec{U}_j^u \right) \tag{2.67}$$

$$\vec{U}_j \quad \leftarrow \quad \Gamma^T\, \vec{U}_i \tag{2.68}$$

Another quantities that require appropriate treatment for periodic boundaries are gradients. In STAMPS, the nodal gradients are calculated using a Green-Gauss formula i.e. integration over the boundary of the closed control volumes - see Section 2.3.3. In order to obtain a correct integral for periodic nodes with a modified control volume as shown in Figure 2.17 (Eq. 2.63), the integrals of lower and upper periodic nodes have to be combined appropriately.

For the translational periodic, only the summation of gradients at node i and j has to be carried out. For rotational periodic boundary condition all vectors have to be first rotated before they can be added - see Eqs. 2.69 and 2.70. This applies to gradients of scalar variables such as density, pressure, and SA-variable.

$$\nabla\phi_i \quad \leftarrow \quad \left(\nabla\phi_i^l + \Gamma_\nabla \; \nabla\phi_j^u\right) \tag{2.69}$$

$$\nabla\phi_j \quad \leftarrow \quad \Gamma_\nabla^T \; \nabla\phi_i \tag{2.70}$$

A 3×3 rotation matrix Γ_∇ is defined in Eq. 2.71

$$\Gamma_\nabla = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\alpha) & -\sin(\alpha) \\ 0 & \sin(\alpha) & \cos(\alpha) \end{bmatrix} \tag{2.71}$$

Velocity gradient, on the other hand, is a 3×3 matrix, and it requires rotation of partial derivatives with respect to $y, z - coordiantess$ as well as partial derivatives related to $v, w - velocities$. This can be obtained using the rotation matrix Γ_∇ as shown in Eqs. 2.72 and 2.73.

$$\nabla\vec{U}_{v,i} \quad \leftarrow \quad \left(\nabla\vec{U}_{v,i}^l + \Gamma_\nabla^T \; \nabla\vec{U}_{v,j}^u \; \Gamma_\nabla\right) \tag{2.72}$$

$$\nabla\vec{U}_{v,j} \quad \leftarrow \quad \Gamma_\nabla \; \nabla\vec{U}_{v,i} \; \Gamma_\nabla^T \tag{2.73}$$

Next, residuals at the corresponding periodic nodes (lower - i, and upper j) have to be combined as shown in Eqs. 2.74 and 2.75, where the rotation matrix Γ is defined as in Eq. 2.62. Note that the periodic treatment of residuals is performed before any volume sources are added. Furthermore, for the multigrid source term, the interpolated fine grid residuals (first term in Eq. 2.95) are treated as flow variables (Eqs. 2.67 and 2.68) before the summation takes place. This operation is required to prevent inconsistent values at the lower and upper periodic patches after the interpolation.

$$\vec{R}_i \quad \leftarrow \quad \left(\vec{R}_i^l + \Gamma \; \vec{R}_j^u\right) \tag{2.74}$$

$$\vec{R}_j \quad \leftarrow \quad \Gamma^T \; \vec{R}_i \tag{2.75}$$

Finally, the treatment of the approximate Jacobian has to be applied (required

for implicit solvers). In STAMPS, there are two preconditioner type options for implicit solver a) the preconditioner based on the approximate block-diagonal Jacobian of the 1^{st}-order spatial discretisation scheme (0^{th}-order Jacobian) - block-Jacobi solver, and b) the preconditioner based on the exact Jacobian of the 1^{st}-order spatial discretisation scheme (1^{st}-order Jacobian) - JT-KIRK solver. For the block-Jacobi, the periodic corrections can be done easily with few adjustments. Block matrices B that correspond to the lower and upper periodic nodes can be summed up and equalised for the translational periodic cases. For the rotational periodicity, the block matrix from the lower periodic node has to be first rotated before the summation is done, and the obtained sum rotated back (Eq. 2.76). The rotation matrix from Eq. 2.62 is used for this purpose.

$$\begin{aligned} B_{i,i} &\leftarrow B^{l}_{i,i} + \Gamma^{T}\ B^{u}_{j,j}\ \Gamma \\ B_{j,j} &\leftarrow \Gamma\ B_{i,i}\ \Gamma^{T} \end{aligned} \tag{2.76}$$

The treatment required for the 1^{st}-order Jacobian (JT-KIRK solver) is simple when the full matrix (including zero terms) is stored in memory. In practical implementation (including in STAMPS) this is never the case, as the memory requirements are too high. Only non-zero terms are stored using a standard compressed row storage (CRS) format. Periodic correction for the 1^{st}-order Jacobian can be done without any adjustments in CRS format for all Jacobian entries related to nodes that lie on the periodic patches. However, for the entries that come from interior nodes (internal node connected with periodic patch), the appropriate rows of a CRS matrix have to be extended. That requires modification of the CRS matrix connectivities and size. An example from Figure 2.16 is used to present the required changes in a transparent way. Periodic nodes (1) and (9) and all nodes/edges required for a 1^{st}-order Jacobian are included in the figure.

The Jacobian matrix that corresponds to the case without periodic patches is presented in Eq. 2.77, whereas the matrix after the treatment of periodic pairs

and appropriate extensions of the original Jacobian is shown in Eq. 2.78.

$$\frac{\partial R}{\partial W} = \begin{bmatrix} B^l_{1,1} & B^l_{1,2} & B^l_{1,3} & B^l_{1,4} & B^l_{1,5} & [0] & [0] & [0] & [0] & [0] & . & . & . \\ & . & & & & & & & & & & & \\ & & . & & & & & & & & & & \\ & & & . & & & & & & & & & \\ & & & & . & & & & & & & & \\ & & & & & . & & & & & & & \\ & & & & & & . & & & & & & \\ [0] & [0] & [0] & [0] & [0] & B^u_{9,6} & B^u_{9,7} & B^u_{9,8} & B^u_{9,9} & B^u_{9,10} & . & . & . \\ & & & & & & & & & . & & & \\ & & & & & & & & & & . & & \\ & & & & & & & & & & & . & \\ & & & & & & & & & & & & B_{n,n} \end{bmatrix} \tag{2.77}$$

$$\frac{\partial R}{\partial W_p} = \begin{bmatrix} B_{1,1} & B_{1,2} & B_{1,3} & B_{1,4} & B_{1,5} & [0] & [0] & [0] & [0] & B_{1,10} & . & . & . \\ & . & & & & & & & & & & & \\ & & . & & & & & & & & & & \\ & & & . & & & & & & & & & \\ & & & & . & & & & & & & & \\ & & & & & . & & & & & & & \\ & & & & & & . & & & & & & \\ [0] & [0] & [0] & B_{9,4} & [0] & B_{9,6} & B_{9,7} & B_{9,8} & B_{9,9} & B_{9,10} & . & . & . \\ & & & & & & & & & . & & & \\ & & & & & & & & & & . & & \\ & & & & & & & & & & & . & \\ & & & & & & & & & & & & B_{n,n} \end{bmatrix} \tag{2.78}$$

Required treatments of the base Jacobian matrix (Eq. 2.77) are summarised below.

1. All matrix entries related to nodes that lies at the periodic patches (Figure 2.16) have to be updated. The treatment presented in Eq. 2.76 has to be applied to the following block matrix pairs: $(B^l_{1,1} - B^u_{9,9})$, $(B^l_{1,2} - B^u_{9,6})$, $(B^l_{1,3} - B^u_{9,8})$, $(B^l_{1,5} - B^u_{9,7})$.

2. New entries have to be added in the Jacobian matrix. These entries cor-

respond to the derivatives of R_1 and R_9 with respect to the interior nodes (4) and (10) from Figure 2.16. The correction described in Eq. 2.76 has to be completed for pairs $(B^l_{1,4} - B^u_{9,4})$ and $(B^l_{1,10} - B^u_{9,10})$ - note that matrix $B^u_{1,10}$ and $B^u_{9,4}$ were zero matrices before the periodic correction.

In the current implementation the second adjustment is not performed.

2.4 Time marching for steady-state flows

STAMPS uses one of the most commonly known method for solving steady-state flow equations, that is, the so-called *method of lines* which is based on the separate discretisaiton in space and time [5]. Starting with the initial guess (e.g. setting flow variables to freestream conditions) the solution is advanced iteratively using pseudo-time marching method until the spatial residuals of the system of flow equations (Eq. 2.7) converge to zero - in practice to computer precision.

Using the notation from Section 2.3 i.e. denoting the space integral using residual vector $\vec{R}$ (Eq. 2.7) the RANS system of equations can be written in the simple form of Eq. 2.79.

$$\frac{\partial}{\partial t}\int_{\Omega} \vec{W} d\Omega = -\vec{R}(\vec{W}) \tag{2.79}$$

When the steady-state solution on a static grid is of interest, the time term in Eq. 2.79 can be discretised using a simple integral and forward difference formula for each control volume Ω_i as presented in Eq. 2.80, which is a basis for **explicit solvers**.

$$\Delta\vec{W}_i^n = -\frac{\Delta t_i^n}{\Omega_i}\vec{R}_i^n \tag{2.80}$$

The solution update is $\Delta\vec{W}_i^n = \vec{W}_i^{n+1} - \vec{W}_i^n$, and the space integral (residual $\vec{R}$) is discretised as described in Section 2.3. The subscript i indicates the particular control volume, and the superscripts n and $n+1$ indicate time levels, where n is the current one. The time step size in the explicit schemes is limited by the stability criteria discussed later.

The basis for **implicit solvers** is presented in Eq. 2.81.

$$\Delta\vec{W}_i^n = -\frac{\Delta t_i}{\Omega_i}\vec{R}_i^{n+1} \tag{2.81}$$

The nonlinear residual at time step $n+1$ cannot be obtained explicitly, however, it can be approximated using a Taylor series, neglecting higher-order terms - Eq. 2.82.

$$\vec{R}_i^{n+1} \approx \vec{R}_i^n + \left(\frac{\partial\vec{R}}{\partial\vec{W}}\right)_i^n \Delta\vec{W}_i^n \tag{2.82}$$

After substituting Eq. 2.82 into Eq. 2.81 and further rearranging, the implicit scheme can be written using Eq. 2.83. A much larger time step size can usually be used for an implicit scheme as compared to a explicit technique, which is the key advantage of the former. However, it comes at the price of higher memory requirements and implementation complexity.

$$\left[\frac{\Omega_i}{\Delta t_i^n} + \left(\frac{\partial\vec{R}}{\partial\vec{W}}\right)_i^n\right]\Delta\vec{W}_i^n = -\vec{R}_i^n \tag{2.83}$$

Eq. 2.83 can be written in short form as shown in Eq. 2.84, where $\mathbf{A}_i^n$ is used to denote the system matrix.

$$\mathbf{A}_i^n\Delta\vec{W}_i^n = -\vec{R}_i^n \tag{2.84}$$

The system matrix (A_i^n) becomes equivalent to the Jacobian of RANS system of equations ($\frac{\partial R}{\partial W}$) as the term $\frac{\Omega_i}{\Delta t_i}$ (Eq. 2.83) converges to zero with the time step rising to infinity ($\Delta t \to +\infty$). The Newton step is then achieved. In practice the finite time step size has to be used at the initial convergence state, when the solution is far from the stationary point and high non-linearities of the RANS system are tackled. Furthermore, in most CFD codes the approximate form of the exact Jacobian of the discrete system of equations ($\frac{\partial R}{\partial W}$) is usually used due to the high memory requirements of the full Jacobian.

There are three steady-state solvers available in STAMPS: 1) explicit, 2) implicit block-Jacobi, and 3) implicit JT-KIRK. Details on each solver are presented below.

Explicit Runge-Kutta (RK) with local time stepping (EX)

As the single-stage explicit schemes are not practically useful for a CFD solver the multi-stage schemes were developed [65]. By default, the 3-stage Runge-Kutta ($N_{rk} = 3$) is used in STAMPS - Eq. 2.85, where the superscript in parentheses denote RK stage. The coefficients α used for each Runge-Kutta stage are $\alpha_1 = 0.1918$, $\alpha_2 = 0.4929$, $\alpha_3 = 1.0$. The subscript i and superscript n are skipped for clarity. Vectors of residuals $\vec{R}^{(0)}$, $\vec{R}^{(1)}$, $\vec{R}^{(2)}$ corresponds to residuals calculated for each RK stage.

$$
\begin{aligned}
\vec{W}^{(0)} &= \vec{W}_i^n \\
\vec{W}^{(1)} &= \vec{W}^{(0)} - \alpha_1 \frac{\Delta t^n}{\Omega} \vec{R}^{(0)} \\
\vec{W}^{(2)} &= \vec{W}^{(1)} - \alpha_2 \frac{\Delta t^n}{\Omega} \vec{R}^{(1)} \\
\vec{W}^{(3)} &= \vec{W}^{(2)} - \alpha_3 \frac{\Delta t^n}{\Omega} \vec{R}^{(2)} \\
\vec{W}_i^{n+1} &= \vec{W}_i^{(3)}
\end{aligned}
\tag{2.85}
$$

The time step Δt is calculated for each control volume independently - Eq. 2.86, where the Courant-Friedrichs-Lewy condition is set to $CFL = 0.69$, Λ_c and Λ_v are convective and viscous spectral radii, and constant c_v is set to 1.0, which is a usual choice for second-order upwind discretisaiton with 3-stage Runge-Kutta scheme [139, 140]. The presented CFL condition (Eq. 2.86) means that for the explicit scheme (Eq. 2.80) the time step should be equal to or smaller than the time required to transport information across the control volume.

$$
\Delta t_i^n = CFL \frac{\Omega_i}{\Lambda_c^n + c_v \Lambda_v^n} \tag{2.86}
$$

Block-Jacobi with Runge-Kutta (BJ)

The RK update procedure (Eq. 2.85) holds; however, at each stage the term $\left(\frac{\Delta t}{\Omega}\vec{R}\right)$ is replaced with the update step $\Delta\vec{W}_i^n$. The update step results from a linear system solve defined in Eq. 2.84, where an approximate block-diagonal Jacobian system matrix is used ($\texttt{diag}\left(\frac{\partial R}{\partial W}\right)_i^n$). This approximate Jacobian is referred to as 0^{th}-order Jacobian that is constructed based on the approximate

Jacobian of the 1^{st}-order spatial discretisaiton scheme. The solver procedure is identical to the one presented in Algorithm 1; the only modification comes from the accuracy of Jacobian approximation.

Jacobian-Trained Krylov-Implicit-Runge-Kutta (JT-KIRK)

JT-KIRK [18] is the default solver that uses Jacobian of the 1^{st}-order spatial discretisaiton scheme. This approximate form of Jacobian will be referred to as the 1^{st}-order Jacobian $\left(\frac{\partial R}{\partial W}\right)^{1st}$.

In a general case, the Jacobian matrix $\frac{\partial R}{\partial W}$ can be obtained automatically using the algorithmic differentiation tool Tapenade [40, 99]. The differentiation can be applied to the residual function that accumulates all fluxes and source terms (see Eq. 2.8) with respect to the vector of conservative variables. In STAMPS, the Jacobian is assembled using manually coded edge and boundary loops and the Tapenade tool is used only to differentiate the core flux functions of the residual - a semi-automatic approach. This method allows Jacobian accumulation time to be reduced as compared to a pure Tapenade approach, and at the same time keeps some level of automation. Whenever the core flux functions are modified, no manual adjustments are required to obtain the new Jacobian matrix.

Calculating an exact Jacobian of the STAMPS's nominal spatial discretisation scheme (2^{nd}-order) would require applying Tapenade differentiation tool to the residual function including procedures for 2^{nd}-order solution reconstruction and gradient calculation. The exact Jacobian of the 2^{nd}-order discretisation scheme is currently not available in STAMPS due to implementation challenges and extensive additional memory requirements caused by wide mesh stencil [122]. The non-zero entries in the exact Jacobian for an example node i require derivative for the node i itself (diagonal term), nearest neighbours of node i, and neighbours of the neighbours of node i (related to gradient calculation). In the case of block-Jacobi solver, the 0^{th}-order Jacobian (block-Jacobian) is calculated by applying Tapenade tool on the parts of the STAMPS's code that relates to 1^{st}-order discretisation scheme and keeping only the diagonal terms of the resultant Jacobian matrix. Achieving 1^{st}-order Jacobian for the JT-KIRK solver requires differenti-

ation of the 1^{st}-order spatial discretisation scheme and storing the diagonal terms of the matrix as well as off-diagonal terms related to the nearest neighbours of an example node i. More details on Jacobian assembly in STAMPS are presented in [18, 122].

The steady solver framework is presented in Algorithm 1. It covers all flow solver types available in STAMPS. The Jacobian $\mathbf{A}$ and preconditioner $\mathbf{P}$ are evaluated only once per RK cycle. The variable $\bar{\mathcal{M}}$ stands for the mesh metrics i.e. edge weights $\vec{s}_{ij}$, boundary weights $\vec{s}_{i_b}$, volumes Ω, and coordinates $\vec{X}$. The multigrid source term $\vec{Q}_{i,MG}$ is explained in Section 2.6.

Algorithm 1 Flow solver in STAMPS

1: **procedure** SOLVER
2: $\Delta t_i \leftarrow$ `localtimestep` $\left(\vec{U}_i^n, \bar{\mathcal{M}}\right)$
3: $\mathbf{A}_i^n \leftarrow$ `jacobian` $\left(\vec{U}_i^n, \bar{\mathcal{M}}, \Delta t_i, CFL\right)$
4: $\mathbf{P}_i^n \leftarrow$ `preconditioner` $\left(\mathbf{A}_i^n\right)$
5: $\vec{U}_i^{(0)} \leftarrow \vec{U}_i^n$
6: **do** $i_{rk} = 0,\ N_{rk} - 1$
7: $\vec{R}_i^{(i_{rk})} \leftarrow$ `residual` $\left(\vec{U}_i^{(i_{rk})}, \bar{\mathcal{M}}\right) + \vec{Q}_{i,MG}$
8: $\Delta\vec{W}_i^{(i_{rk})} \leftarrow$ `linearsolver` $\left(\mathbf{A}_i^n, \mathbf{P}_i^n, \vec{R}_i^{(i_{rk})}\right)$
9: $\vec{U}_i^{i_{rk}+1} \leftarrow$ `updateflow` $\left(\vec{U}_i^{(0)}, \Delta\vec{W}_i^{(i_{rk})}, \alpha_{i_{rk}+1}\right)$
10: **end do**
11: $\vec{U}_i^{n+1} \leftarrow \vec{U}_i^{N_{rk}}$

Depending on solver choice (EX/BJ/JT-KIRK) some of the functions differ internally, as with, `jacobian`() - line 3, `preconditioner`() - line 4, and `linearsolver`() - line 8. Each line of the Alg. 1 is described in detail below. Point IDs in the list correspond to the line numbers.

2. Calculate local time step (Eq. 2.86).

3. Use an approximate system matrix $\mathbf{A}$ as defined in Eq. 2.84. Depending on the solver type the matrix is approximated differently:

- EX: $\mathbf{A} = \texttt{diag}\left(\frac{\Omega}{\Delta t}\right)$, diagonal matrix.
- BJ: $\mathbf{A} = \left[\texttt{diag}\left(\frac{\Omega}{\Delta t}\right) + \texttt{diag}\left(\frac{\partial R}{\partial W}\right)\right]$, block-diagonal matrix.
- JT-KIRK: $\mathbf{A} = \left[\texttt{diag}\left(\frac{\Omega}{\Delta t}\right) + \frac{\partial R}{\partial W}^{1^{st}}\right]$, Jacobian of the 1^{st}-order spatial discretisation scheme $\left(\frac{\partial R}{\partial W}\right)^{1^{st}}$.

4. Compute preconditioner of the system matrix A:

   - EX: $\mathbf{P} = \mathbf{A}^{-1} = \texttt{diag}\left(\frac{\Delta t}{\Omega}\right)$. Preconditioner is obtained using direct inversion of diagonal matrix $\mathbf{A}$.
   - BJ: $\mathbf{P} = \mathbf{A}^{-1} = \left[\texttt{diag}\left(\frac{\Delta t}{\Omega}\right) + \texttt{diag}\left(\frac{\partial R}{\partial W}\right)\right]^{-1}$. Preconditioner is computed using direct inversion of the system matrix $\mathbf{A}$. Gaussian elimination is used for this purpose.
   - JT-KIRK: the incomplete lower upper factorisation $ILU(0)$ is used for the preconditioner, which is an approximate of the system matrix i.e. $\mathbf{A} \approx \mathbf{LU}$, where $\mathbf{L}$ and $\mathbf{U}$ are lower and upper triangular matrix, respectively. No direct inversion is done.

5. Initialise the RK loop.

7. Compute nonlinear residual for the current flow vector $\vec{U}_i^{i_{rk}}$. For the multigrid solver the appropriate source term has to be added ($\vec{Q}_{i,MG}$) - see Section 2.6 for details.

8. Calculate solution update. Depending on the choice of the solver the update for a single RK step is calculated as follows:

   - EX: the update is presented in Eq. 2.87

$$\Delta\vec{W}_i^{(i_{rk})} = \alpha_{i_{rk}} \frac{\Delta t^n}{\Omega_i} \vec{R}_i^{(i_{rk})} \tag{2.87}$$

   - BJ: the update is shown in Eq. 2.88. Preconditioner matrix is calculated as explained in point (4) for BJ solver.

$$\Delta\vec{W}_i^{(i_{rk})} = \alpha_{i_{rk}} P^n \vec{R}_i^{(i_{rk})} \tag{2.88}$$

- JT-KIRK: as the direct inversion of the 1^{st}-order system matrix $\mathbf{A}$ is not feasible for large system of equations, the GMRES linear solver [68] with the $ILU(0)$ preconditioner is used to obtained the solution update $\Delta\vec{W}_i^{(i_{rk})}$ for JT-KIRK scheme.

9. Update solution variables. Inside the $update flow()$ function, first obtain the conservative variables $\vec{W}^{(0)}$ for the flow state variables $\vec{U}^{(0)}$; next update the vector $\vec{W}^{(i_{rk}+1)}$ using step size $\Delta\vec{W}^{(i_{rk})}$; then evaluate the flow vector $\vec{U}^{(i_{rk}+1)}$ using correlations between conservative and flow variables.

   Note that internally STAMPS stores flow variables vector $\vec{U}$ not the vector of conservative variables $\vec{W}$. Hence, the appropriate transformation is applied inside `updateflow()` function of the update step.

As already mentioned in this section, implicit schemes like JT-KIRK often require a finite time step size Δt at the initial convergence stage, when high non-linearities of the RANS system of equations are tackled. The finite time step increases the diagonal dominance of the system matrix $\mathbf{A}$ and allow to evolve the solution slower in a finite pseudo-time. This helps to stabilise the Newton-type solver which is sensitive to an initial guess. The step size Δt - Eq. 2.86, is usually controlled by the user defined CFL number. Ideally, a small initial CFL number should be used and then increased as the solution progresses towards the stationary point. To fulfil this task, this research implemented the automatic CFL-adjustment technique based on the line search algorithm introduced by Michalak [20]. Several other solver enhancements, e.g. Jacobian re-computation control, were introduced to increase the robustness of the solver. All the improvements are summarised in Chapter 4.

The unsteady solver is also available in STAMPS. Physical time is discretised using implicit dual time-stepping approach with backward differentiation formula 2 (BDF2) [66]. The theory as well as the implementation details can be found in Section 6.3 in [102].

2.5 Interpolation operators

Interpolation operators allow to transfer variables between different grids. A restriction operator is used to interpolate variables from fine to coarse meshes, and a prolongation operator is used to transfer variables in the opposite direction (from coarse to fine grids). Interpolation operators are required for geometric multigrid solvers and for truncation error estimation. Both functionalities are available in STAMPS (as introduced in Section 2.1), and both impose specify requirements on the order of accuracy of the interpolation operators. This section describes interpolation operators available in STAMPS and summarises the result of a rigorous accuracy test to confirm that the requirements set by multigrid solver and truncation error estimations are met. The details on multigrid solver and error estimation are provided in Sections 2.6 and 5.2.2, respectively.

The effective multigrid solver method requires the combined accuracy of the prolongation and restriction operators to be higher than the order of equations being discretised [92] (2 for the Navier-Stokes equations). This means that at least one of the interpolations operators has to provide 2^{nd}-order accuracy. The accurate truncation error estimation method requires both operators to be at least consistent with the design order of accuracy of the discretisation scheme (2^{nd}-order in STAMPS).

The notation used for the interpolation operators is as follows:

- restriction operator $\mathcal{I}_h^H$
- prolongation operator $\mathcal{I}_H^h$

The lowercase h refers to the fine discrete space, whereas the uppercase H to the coarse discrete space. Hence, the operator $\mathcal{I}_h^H$ allow to transfer variables from the fine grid (fine discrete space) to the coarse grid (coarse discrete space). The subscript in the interpolation operator denotes the discrete source-space, and the superscript the discrete target-space (where the variables are interpolated to).

Two interpolation methods are available in STAMPS, i.e. gradient-based and coefficient-based, where for the latter the interpolation coefficients are obtained using a minimum-norm solution within a local stencil [122]. In both cases the interpolation stencil for each fine grid node is defined by the coarse element that contains given fine grid node - see a simple 2D example in Figure 2.18. Hence, the prolongation operator uses gradient/coefficients stored at nodes which form the coarse element. The stencil for restriction operator is essentially a transposed prolongation operator's stencil (by duality). This stencil/connectivity is provided by the tool *hip*[23] [37].

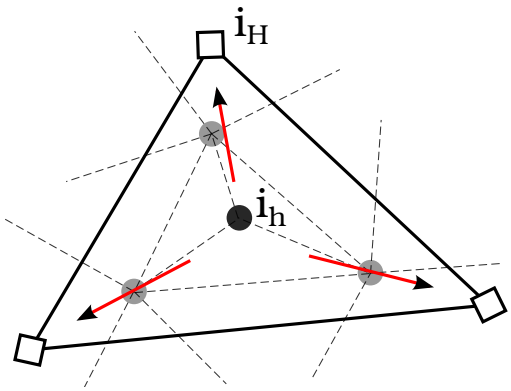


Figure 2.18: Multigrid connectivity - dashed line - fine grid (i_h), solid line - coarse grid (i_H)

The minimum norm interpolation operators were implemented in STAMPS by the former PhD student Shenren Xu and are described in detail in [122]. The gradient-based operators implemented for this research are described in more detail below.

The gradient-based interpolation uses a cell-based gradient which is exact for linear fields - see Section 2.3.5. The prolongation of a field ϕ is performed as shown in Eq. 2.89. The variable $N_H(i)$ stands for the number of coarse nodes i_H within the interpolation stencil of the fine node i_h. And $\vec{r}_H^{\,h}$ as shown in Eq. 2.90, is a vector that points from the coarse grid node i_H to fine grid node i_h. Restriction of the field ϕ is done correspondingly, using the transposed prolongation stencil.

$$\phi_{i_h} = \sum_{i_H=1}^{N_H(i)} \left(\Omega_{i_H} \left(\phi_{i_H} + \nabla \phi_{i_H} \cdot \vec{r}_H^{\,h} \right) \right) \Big/ \sum_{i_H=1}^{N_H(i)} \Omega_{i_H} \tag{2.89}$$

[23] A package to manipulate unstructured computational grids

$$\vec{r}_H^{\,h} = \vec{X}_{i_h} - \vec{X}_{i_H} \tag{2.90}$$

To test the accuracy of interpolation operators, a mesh convergence study was performed using a simple cube domain with a general mixed grid type. A set of 7 grids was used, where each refined stage had a roughly halved characteristic mesh size. The nonlinear function field that corresponds to the density-manufactured-solution from Eq. 2.91 was used (equation constants are provided in Appendix A.5). To test the prolongation accuracy, th presented field was first evaluated on coarser grids and then prolonged to fine mesh respectively. Similarly the procedure was applied to test the accuracy of restriction operators.

$$\rho(x,y,z) = \rho_0 + \rho_x \sin\left(\frac{\alpha_{\rho x}\pi x}{L}\right) + \rho_y \cos\left(\frac{\alpha_{\rho y}\pi y}{L}\right) + \rho_z \sin\left(\frac{\alpha_{\rho z}\pi z}{L}\right) \tag{2.91}$$

Next, the L_2-norm of the interpolation error was calculated based on the Eq. 2.92 which is written for the prolonged field ϕ_h^H. The indices in variable ϕ_h^H refer to the quantity ϕ on fine discrete space h interpolated from coarse space H.

$$||\epsilon_h^H||_{2,N} = \sqrt{\left(\sum_{i=1}^{N}\left(\phi_{h,i} - \phi_{h,i}^H\right)^2\right) \;/\; N} \tag{2.92}$$

The characteristic (effective) mesh size h_e is obtained according to Eq. 2.93, where N is the number of degrees of freedom (mesh nodes fro node-centred scheme).

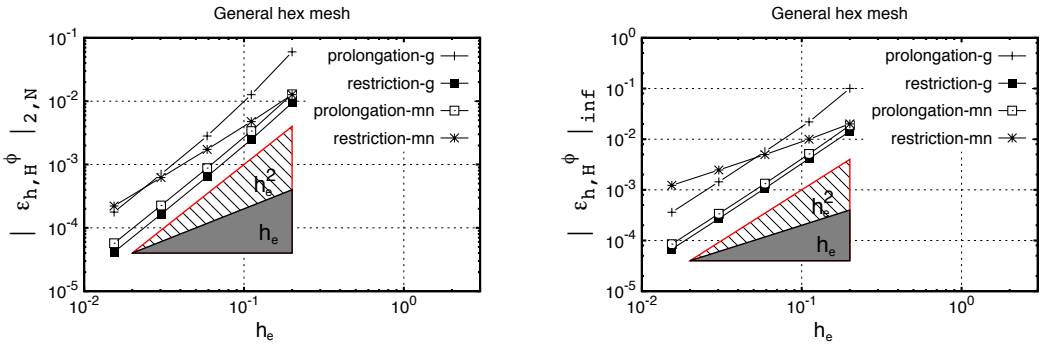
$$h_e = \left(\frac{1}{N}\right)^{\frac{1}{3}} \tag{2.93}$$

The effective mesh size can be seen as a non-dimensional edge length of a discrete element forming a computational domain with a unit volume and uniformly distributed nodes. In principle, in a p^{th}-order accurate method (e.g. $p = 2$ for 2^{nd}-order) the example L_2-norm of the error (Eq. 2.92) is expected to reduce at a rate of h_e^2 (reduction of h_e by factor 2 should lead to reduction of errors by factor 4). The variable h_e is frequently used in the literature on code verification and error analysis (see Roy [69, 70], or Diskin [63, 120]) and is also adopted in this work. The words effective mesh size, characteristic mesh size, and equivalent

mesh size will be used interchangeably in this work and will always refer to the variable h_e.

The results confirmed that the gradient-based interpolation operators implemented in STAMPS provide 2^{nd}-order accurate transfer of the solution fields between meshes as shown in Figure 2.19 (*prolongation–g*, *restriction–g*). The errors converge with a h_e^2-slope when inspecting both the L_2-norm and the L_∞-norm, where the latter provides a more rigorous answer as it shows the worst converging interpolation errors within the entire domain.

The minimum-norm-based prolongation operator provides a 2^{nd}-order accuracy, whereas restriction is $1st$-order accurate as presented in Figure 2.19 (*prolongation–mn*, *restriction–mn*). The reduced order of accuracy of restriction was expected as in the current implementation the same (but transposed) set of prolongation coefficients was used. This property, i.e. $\mathcal{I}_h^H = \left(\mathcal{I}_H^h\right)^T$ is important for the efficiency of geometric multigrid solvers [92]. The interpolation accuracy requirement for an effective geometric multigrid solver is still satisfied for RANS system of equations, i.e. $m_r + m_p > m_{eq},\ (3 > 2)$.



(a) $L_2 - norm$ (b) $L_\infty - norm$

Figure 2.19: Interpolation accuracy test

The minimum norm transfer operators are used for a geometric multigrid solver (Section 2.6), whereas the consistently 2^{nd}-order gradient-based interpolation is used for truncation error estimation explained more in Section 5.2.2.

It should be also noted that for highly-curved domains there can be fine grid nodes which are not contained in any coarse element. In such a case, the nearest

coarse element is used for interpolation. For the gradient-based operator, the interpolation remains 2^{nd}-order, which is not the case for minimum-norm-based operators [122].

2.6 Geometric multigrid

Geometric-Multi-Grid (GMG) is a powerful convergence acceleration technique. It was first introduced in the 1960s by Fedorenko [123] and Bakhvalov [124] and further advanced e.g. by Brandt [125, 126] or Jameson [127, 128]. In this method, the solution of the governing equations is obtained using a series of successively coarsened meshes, which are used to drive the solution on the finest grid faster to the steady-state. The key advantages come from the fact that larger time steps can be used on coarse grids, and the low-frequency components of the solution errors can be reduced more efficiently on the coarse grid. The latter is a crucial feature of multigrid technique as the explicit and implicit solvers are known to damp effectively mainly high-frequency error modes. Hence, the multigrid methodology combined with any of the time-stepping schemes (EX, BJ, JT-KIRK) described in Section 2.4 lead to a very efficient solving technique. Furthermore, the coarse grids can also be used for estimation of the truncation error to drive the mesh adaptation process - see Chapter 5.

The advantages of the GMG technique comes at a price of larger implementation effort and a need to generate a series of coarse grids - at least one. The element-collapsing algorithm of the tool *hip*, which was briefly introduced in Section 2.5, is used for the mesh coarsening [37].

In STAMPS, the Full Approximation Storage method (FAS) is used, where re-discretisation is performed for the coarse grids. As the coarse grids discretisation accuracy has no effect on the fine grid solution accuracy, the 1^{st}-order scheme is employed on the coarse meshes. This decreases computational effort, increases robustness, and provides better high-frequency damping properties. Additionally, the 1^{st}-order scheme is less sensitive to mesh quality, which can be compromised on coarse grids by the element-collapsing process.

The key steps of the multigrid FAS cycle are introduced below, where the

subscript h is used to denote quantities related to fine grid level, and uppercase H is indicating coarse grid variables and operators. For example, $\vec{U}_h$ is a solution vector on the fine grid and $\vec{U}_H$ on the coarse grid. Both subscripts stand for the characteristic mesh size.

1. Pre-smoothing: perform a single solver iteration on the finest grid h. The updated flow vector is thus obtained i.e. $\vec{U}_h^{n+1}$.

2. Restriction: transfer of quantities to coarse grid H. The quantities to be restricted are the residuals $\vec{R}_h$ and the solution vector $\vec{U}_h$. The appropriate interpolation operators are used for this purpose, and the procedure can be written for the solution vector as presented in Eq. 2.94.

$$\vec{U}_H^0 = \mathcal{I}_h^H \vec{U}_h^{n+1} \tag{2.94}$$

   The current residual is evaluated on the coarse grid, i.e. $\vec{R}_H^0 = \vec{R}_H(\vec{U}_H^0)$, and then subtracted from the restricted fine grid residual $\widehat{\mathcal{I}}_h^H \vec{R}_h^{n+1}$ to form the so-called multigrid forcing term $\vec{Q}_{MG}$, Eq. 2.95. This guarantees that the solution on the coarse mesh depends on the fine grid residual.

$$\left(\vec{Q}_{MG}\right)_H = \widehat{\mathcal{I}}_h^H \vec{R}_h^{n+1} - \vec{R}_H^0 \tag{2.95}$$

   Note that the operators used for restriction of solution $\mathcal{I}_h^H$ and residuals $\widehat{\mathcal{I}}_h^H$ are different. In the case of residual restriction the operator has to ensure a conservative transfer, that is, as the control volume size increases the residual must increase by the same amount.

3. Coarse grid solve: calculate a new solution on the coarse grid H. The smoother call is executed for the coarse grid H as presented in Alg. 1 - replace subscript i with H. The updated solution $\vec{U}_H^{n+1}$ is obtained on the coarse grid.

4. Prolongation: coarse grid solution interpolation and multigrid update on the fine mesh h. The coarse grid correction $\Delta\vec{U}_H^{MG}$ is first calculated as shown in Eq. 2.96.

$$\Delta\vec{U}_H^{MG} = \vec{U}_H^{n+1} - \vec{U}_H^0 \tag{2.96}$$

The coarse grid correction is then interpolated to the fine grid using prolongation operator $\mathcal{I}_H^h$ (see Section 2.5). Finally, the solution on the fine grid is advanced (Eq. 2.97).

$$\vec{U}_h^{MG} = \vec{U}_h^{n+1} + \mathcal{I}_H^h \Delta \vec{U}_H^{MG} \tag{2.97}$$

The most common V-cycle is implemented in STAMPS. Algorithm 2 shows a single multigrid V-cycle for three multigrid mesh levels. The three grids are used to present how the so-called forcing term $\vec{Q}_{MG}$ is cascaded down towards the coarsest grid. The process is repeated every solver cycle until convergence criteria are met. A number of clarifications are set out below.

- Indices h, $2h$, $3h$ ... refer to each consecutive coarse grid level.
- The variable $\bar{\mathcal{M}}$ collects the mesh metrics i.e. edge weights $\vec{s}_{ij}$, boundary weights $\vec{s}_{i_b}$, volumes Ω, and coordinates $\vec{X}$, for each grid level.
- The residual function `residual`() represents the flux accumulation as described in Section 2.3.
- The smoother function `smoother`() (single solver iteration) is presented in Algorithm 1.
- The multigrid source term for the finest grid is zero, $(\vec{Q}_{MG})_h = 0$.
- The nomenclature $\vec{R}_h^n$ refers to the residual evaluated on the grid level h using solution vector from time step n, i.e. $\vec{R}_h(\vec{U}_h^n)$
- The indentation is used in the algorithm to show operations that take place at each grid level. For example, when a given operation is under the label **Level (h)** but it is indented to **Level (2h)**, it means that the quantity from grid **h** is used to obtain the corresponding value on grid **2h**.

Algorithm 2 Multigrid V-cycle in STAMPS

1: **procedure** FAS-CYCLE

2: **Level (h)**

3: $\vec{U}_h^{n+1} \leftarrow \texttt{smoother}\left(\vec{U}_h^n, \bar{\mathcal{M}}_h, CFL_h^n, (\vec{Q}_{MG})_h\right)$

4: $\vec{R}_h^{n+1} \leftarrow \texttt{residual}\left(\vec{U}_h^{n+1}, \bar{\mathcal{M}}_h\right) + (\vec{Q}_{MG})_h$

5: $\vec{U}_{2h}^0 \leftarrow \mathcal{I}_h^{2h}\vec{U}_h^{n+1}$

6: $\vec{R}_{2h}^0 \leftarrow \texttt{residual}\left(\vec{U}_{2h}^0, \bar{\mathcal{M}}_{2h}\right)$

7: $(\vec{Q}_{MG})_{2h} \leftarrow \widehat{\mathcal{I}}_h^{2h}\vec{R}_h^{n+1} - \vec{R}_{2h}^0$

8: **Level (2h)**

9: $\vec{U}_{2h}^n \leftarrow \vec{U}_{2h}^0$

10: $\vec{U}_{2h}^{n+1} \leftarrow \texttt{smoother}\left(\vec{U}_{2h}^n, \bar{\mathcal{M}}_{2h}, CFL_{2h}^n, (\vec{Q}_{MG})_{2h}\right)$

11: $\vec{R}_{2h}^{n+1} \leftarrow \texttt{residual}\left(\vec{U}_{2h}^{n+1}, \bar{\mathcal{M}}_{2h}\right) + (\vec{Q}_{MG})_{2h}$

12: $\vec{U}_{3h}^0 \leftarrow \mathcal{I}_{2h}^{3h}\vec{U}_{2h}^{n+1}$

13: $\vec{R}_{3h}^0 \leftarrow \texttt{residual}\left(\vec{U}_{3h}^0, \bar{\mathcal{M}}_{3h}\right)$

14: $(\vec{Q}_{MG})_{3h} \leftarrow \widehat{\mathcal{I}}_{2h}^{3h}\,\vec{R}_{2h}^{n+1} - \vec{R}_{3h}^0$

15: **Level (3h)**

16: $\vec{U}_{3h}^n \leftarrow \vec{U}_{3h}^0$

17: $\vec{U}_{3h}^{n+1} \leftarrow \texttt{smoother}\left(\vec{U}_{3h}^n, \bar{\mathcal{M}}_{3h}, CFL_{3h}^n, (\vec{Q}_{MG})_{3h}\right)$

18: $\vec{U}_{2h}^{MG} \leftarrow \vec{U}_{2h}^{n+1} + I_{3h}^{2h}\left(\vec{U}_{3h}^{n+1} - \vec{U}_{3h}^0\right)$

19: **Level (2h)**

20: $\vec{U}_{2h}^n \leftarrow \vec{U}_{2h}^{MG}$

21: $\vec{U}_{2h}^{n+1} \leftarrow \texttt{smoother}\left(\vec{U}_{2h}^n, \bar{\mathcal{M}}_{2h}, CFL_{2h}^n, (\vec{Q}_{MG})_{2h}\right)$

22: $\vec{U}_h^{MG} \leftarrow \vec{U}_h^{n+1} + I_{2h}^h\left(\vec{U}_{2h}^{n+1} - \vec{U}_{2h}^0\right)$

23: **Level (h)**

24: $\vec{U}_h^n \leftarrow \vec{U}_h^{MG}$

As initially stated in Section 2.5, an important aspect of an effective multigrid solver is to satisfy the interpolation accuracy requirements as defined in Eq. 2.98, where m_r and m_p are the order of accuracy of restriction and prolongation.

$$m_r + m_p > m_e \tag{2.98}$$

If this condition is not met the multigrid procedure may result in a stall or even divergence due to the errors introduced in the interpolation step. The variable m_e is the order of the governing equations i.e. $m_e = 1$ for the Euler equations, and $m_e = 2$ for the Navier-Stokes equations. As will be shown in Section 2.5, in STAMPS the restriction operator is 1^{st}-order accurate ($m_r = 1$), and the prolongation is 2^{nd}-order accurate ($m_p = 2$), hence the implementation satisfies condition in Eq. 2.98. See Briggs [92] for more details on the interpolation accuracy requirement.

Finally, to take an advantage of the multigrid technique for solution initialisation purpose, the multigrid start-up (MGS) is implemented in STAMPS. It is a simplified version of full multigrid start-up (FMG) which is simple to implement and allows to obtain a better initial flow state for the finest mesh. Multigrid start-up is executed before starting the main solver. First, a user-defined number of solver iterations are run on the least computationally demanding coarsest grid, then the obtained state is interpolated to the next grid level and used for initialisation there. This cycle is repeated until the finest mesh is reached, and then the V-cycle presented in Algorithm 2 continues. As a result of a better initial flow state the overall solver run-time reductions can be achieved. The MGS is described and tested in Section 4.5.

2.7 Verification and validation

Verification and validation (V&V) of computational tools are very important aspects in engineering simulations. In CFD, for example, V&V activities are helpful in assessing the applicability of the selected mathematical models, correctness of the computer codes and numerical algorithms, and quantifying the numerical accuracy.

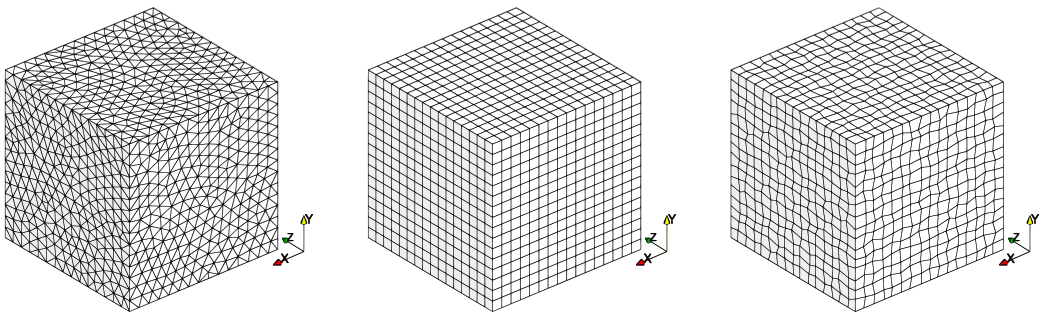
Solver validation allows checking whether the implemented mathematical model predicts the physical behaviour correctly. Usually, a set of experimental data obtained, e.g. in wind tunnels, is used for this purpose. Validation of STAMPS has already been performed by former PhD students in their dissertations - see Christakopoulos [129] and Xu [122] - hence it is not covered in this work.

The role of solver verification is to ensure that there are no coding mistakes (bugs), algorithm inconsistencies, and the numerical errors are converging with an expected slope as the discrete computational space is being refined. The order of accuracy test is considered as one of the most rigorous verification techniques, where the method of manufactured solution (MMS) plays a key role [41]. Details on solver verification techniques and method of manufactured solution are provided in [69, 70].

First, the gradient calculation accuracy test is performed. Next, the overall spatial discretisation accuracy is analysed.

2.7.1 Verification case

The accuracy of gradient computation and the overall spatial discretisation scheme used in STAMPS is performed using three different mesh types i.e. a tetrahedral grid, a regular hexahedral mesh, and a general hexahedral mesh shown in Figure 2.20. All meshes are prepared for the simple cube domain.



(a) Tetrahedral (b) Regular hexahedral (c) General hexahedral

Figure 2.20: Mesh types used for solver verification - figures show grid refinement level 4 from Table 2.7

For each grid, a sequence of seven uniformly refined grids was prepared to perform a mesh convergence study of the error in the computed gradient (see Section 2.7.2), as well as solution error and truncation error to assess spatial discretisation accuracy (Section 2.7.3). A summary information for all meshes used in testing is provided in Table 2.7.

Mesh level	Tet		Hex, Regular/General	
	nodes	h_e	nodes	h_e
1	2.01M	0.00792	2.15M	0.00775
2	0.27M	0.01538	0.27M	0.01538
3	35.9K	0.03030	35.9K	0.03030
4	4.95K	0.05878	4.91K	0.05882
5	725.0	0.11111	729.0	0.11111
6	127.0	0.19898	125.0	0.20000
7	27.00	0.33333	27.00	0.33333

Table 2.7: Meshes used for error convergence study. Regular hex and general hex meshes have identical statistics

Other grid types such as a general prism, pyramid, and mixed grid are not investigated as the results are expected to be the same as for the general hex mesh presented in Figure 2.20c. Note that regular hex and general hex meshes have identical statistics because the general hex meshes were created through a random perturbation of nodes of the corresponding regular hex meshes.

2.7.2 Gradient accuracy

The accuracy and performance of two gradient computation approaches are compared, i.e., edge-based scheme described in Section 2.3.4, and cell-based scheme described in Section 2.3.5.

The procedure used for the gradient accuracy test is outlined below.

1. Define a continuous field ϕ - Eq. 2.99, and derive a continuous expression for its derivative (gradient) as in Eq. 2.100. The nonlinear manufactured solution field for density is used - see Appendix A.5 for constants.

$$\widetilde{\phi}(x,y,z) = \phi_0 + \phi_x \sin\left(\frac{\alpha_{\phi x}\pi x}{L}\right) + \phi_y \cos\left(\frac{\alpha_{\phi y}\pi y}{L}\right) + \phi_z \sin\left(\frac{\alpha_{\phi z}\pi z}{L}\right) \quad (2.99)$$

$$\widetilde{\nabla\phi}(x,y,z) = \begin{bmatrix} \phi_x \ \frac{\alpha_{\phi x}\pi}{L} \ \cos\left(\frac{\alpha_{\phi x}\pi x}{L}\right) \\ -\phi_y \ \frac{\alpha_{\phi y}\pi}{L} \ \sin\left(\frac{\alpha_{\phi y}\pi y}{L}\right) \\ \phi_z \ \frac{\alpha_{\phi z}\pi}{L} \ \cos\left(\frac{\alpha_{\phi z}\pi z}{L}\right) \end{bmatrix} \tag{2.100}$$

2. Evaluate the discrete gradient Eqs. 2.17 and 2.24 of the nonlinear function in Eq. 2.100 sampled discretely at the vertices. The exact gradient $\widetilde{\nabla\phi}$ can be obtained at node $i : (x_i, y_i, z_i)$ using Eq. 2.100.

3. Calculate the error norm for the gradient. To inspect the order of accuracy of the gradient computation, the two following norms are used, namely $L_{1,N}$ and L_∞ defined in Eqs. 2.101 - 2.102. The subscript N in the $L_{1,N}$ norm stands for the field size. Note that a single error norm is obtained for a gradient (vector) by summing error contributions for each direction (x, y, z), hence division by $3N$ in Eq. 2.101.

$$||\epsilon_h^\nabla||_{1,N} = \left(\sum_{i=1}^{N}\sum_{j=1}^{3}\left|\widetilde{\nabla\phi}_i^j - \nabla\phi_{h,i}^j\right|\right) \ / \ (3N) \tag{2.101}$$

$$||\epsilon_h^\nabla||_\infty = \max_{i,j}\left|\widetilde{\nabla\phi}_i^j - \nabla\phi_{h,i}^j\right| \tag{2.102}$$

4. Perform all operations for each of the 7 grids. The characteristic (equivalent) mesh size h_e is obtained according to Eq. 2.103, where N is the number of degrees of freedom.

$$h_e = \left(\frac{1}{N}\right)^{\frac{1}{3}} \tag{2.103}$$

The detailed analysis of gradient calculation accuracy is performed for each tested mesh type. To achieve p^{th}-order exact solution reconstruction, the $(p-1)^{th}$-order polynomial has to be represented exactly in the computational space. This requires the error in gradient calculation to be at least $O(h_e^{p-1})$ i.e. gradient calculation method to be $(p-1)^{th}$-order. As a result, the 2^{nd}-order solution reconstruction ($p = 2$) requires the linear field (1^{st}-order polynomial) to be represented exactly. This can be achieved when the error in the gradient computation method is $O(h_e)$ which can be explained by analysing 1D solution reconstruction

as in Eq. 2.104.

$$\phi(h_e + \delta h_e) \approx \phi(h_e) + \nabla\phi(h_e)\ \delta h_e \tag{2.104}$$

For the gradient accuracy being of the order $O(h_e)$ and δh_e being $O(h_e)$, the multiplied term ($\delta\phi = \nabla\phi(h_e)\ \delta h_e$) becomes of the order $O(h_e^2)$ resulting in a 2^{nd}-order solution reconstruction. Note that increasing the accuracy of the gradient calculation to higher-order is not enough to increase the order of accuracy of the solution reconstruction. The Eq. 2.104 is obtained using Taylor expansion neglecting higher-order terms. This means that by design it can be maximum $O(h_e^2)$ assuming that the $\delta\phi$ is at least $O(h_e^2)$. Higher-order terms of Taylor expansion are required to achieve 3^{rd} and higher-order reconstruction.

The order of accuracy can be investigated through the analysis of the convergence curve slope of the gradient error norm (see Figures 2.21 - 2.24). This should be at least consistent with a slope of h_e^1. The L_1 norm reveals the dominating accuracy over the entire computational domain whereas the infinity norm shows the worst converging errors. For the scheme to be consistently p^{th}-order accurate over the entire computational domain, the error in each discrete point should converge with the h_e^p-slope. The infinity norm is very useful in assessing whether the above is true and the scheme is consistently p^{th}-order. Additionally, the errors are plotted for three regions:

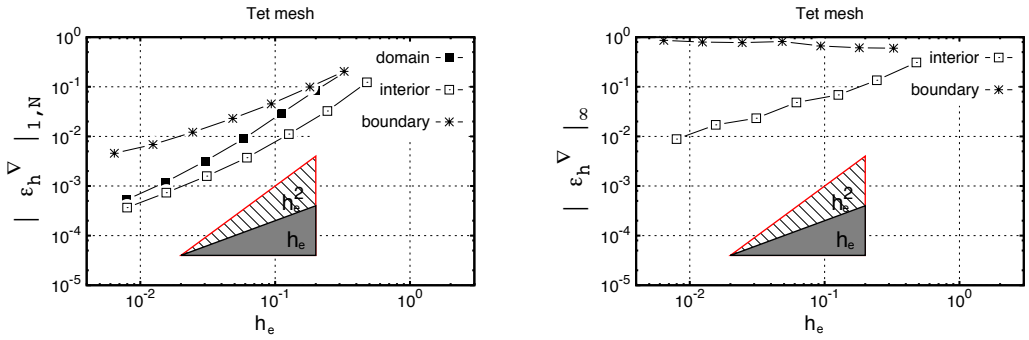
- the entire domain
- only interior nodes
- only boundary nodes.

This allows the location of the degeneration of the accuracy, if any, to be inspected.

Tetrahedral mesh

Figure 2.21a shows that without the boundary correction the order of accuracy of the base edge-based gradient discretisation Eq. 2.17 is between 0^{th}-order and 1^{st}-order. The errors converge with 1^{st}-order slope for the interior nodes (Figure 2.21), whereas the accuracy at the boundary nodes degenerates to 0^{th}-order

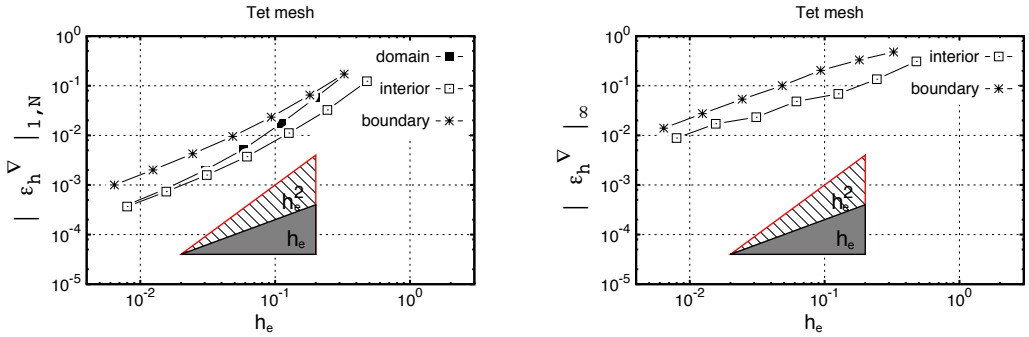
which can be confirmed when analysing the infinity norm (see Figure 2.21b). In order to achieve a consistent 1^{st}-order accurate gradients over the entire domain, the boundary correction described in Section 2.3.4 is required.



(a) $L_{1,N}$ norm (b) L_{∞} norm

Figure 2.21: Gradient accuracy test for the edge-based approach (refer Section 2.3.4)

Figures 2.22a and 2.22b confirm that the proposed cell-based approach gives a consistently 1^{st}-order accurate gradient for the boundary and interior nodes.



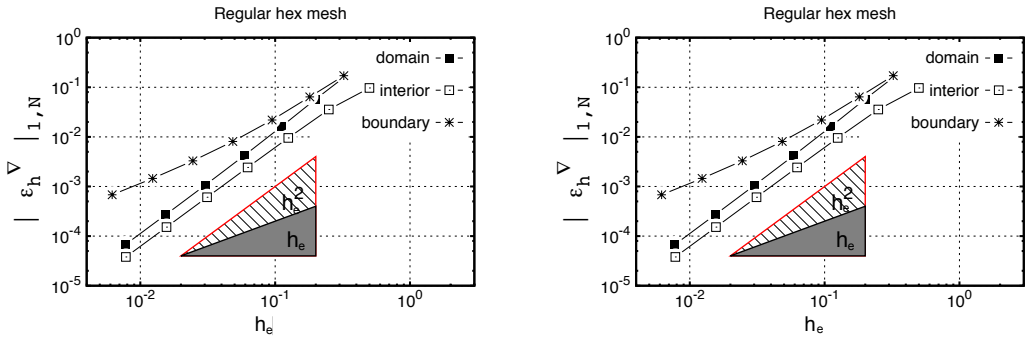
(a) $L_{1,N}$ norm (b) L_{∞} norm

Figure 2.22: Gradient accuracy test for the cell-based approach (refer Section 2.3.5)

Regular hexahedral mesh

For this particular mesh type, a supra-convergence is achieved for both gradient calculation approaches - Figure 2.23. This is a well-known behaviour for regular grids [63], and is a result of a perfectly symmetric stencil that leads to the cancellation of $O(h_e)$ errors in the domain interior. The errors at the boundaries remain $O(h_e)$ for both gradient calculation methods as the stencil is no longer

symmetric there for the vertex-centred scheme. The infinity norm is not shown because the L_1 norm is enough to make a full judgement of the order of accuracy.

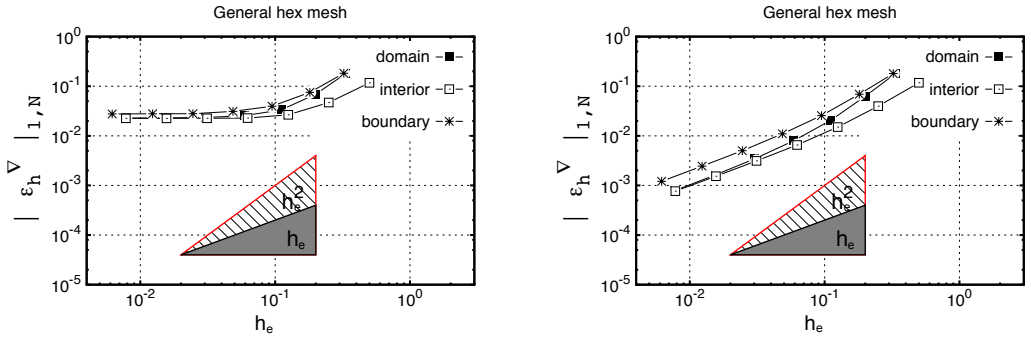


(a) Edge-based (b) Cell-based

Figure 2.23: Gradient accuracy test for a regular hex mesh - $L_{1,N}$ norm

General hexahedral mesh

For the general hexahedral mesh, Figure 2.24a reveals that the gradient calculation accuracy of the standard edge-based scheme is $O(1)$ as expected. The cell-based approach shows consistent 1^{st} accuracy - Figure 2.24b. The same trend is expected to hold for the mixed and general mesh types for both gradient computation methods.



(a) Edge-based (b) Cell-based

Figure 2.24: Gradient accuracy test for a general hex mesh - $L_{1,N}$ norm

For more details on a gradient accuracy testing, an influence of mesh quality, and a comparison of various gradient computation methods, refer to [63, 120, 64].

Performance comparison

Three mesh types are used for the performance comparison - see Table 2.8. A mixed grid of tetrahedrons (88% of all cells) and prisms (12% of all cells) is an addition to tet and hex meshes used in accuracy testing as this combination is very common in practical industrial applications.

Mesh info	Tet	Hex	Tet+prism(10%)
nodes [10^6]	2.01	2.15	1.10
cells [10^6]	11.9	2.19	5.27
edges [10^6]	13.9	6.39	6.65
cells/nodes [-]	5.92	1.26	4.79

Table 2.8: Mesh sizes used for performance comparison (number in millions)

Performance comparison of two methods for gradient computation is shown in Table 2.9. The measurements of run time is based on the average over four consecutive runs. For each run, 30 evaluations of the gradient subroutine for a scalar field ϕ was performed.

Method	Tet		Hex		Tet+Prism(12%)	
edge-based	**1**	(23.5s)	**1**	(8.65s)	**1**	(10.4s)
cell-based	**1.15**	(27.1s)	**0.64**	(5.56s)	**0.88**	(9.19s)

Table 2.9: Performance comparison of edge-based vs cell-based gradient computation - normalised and absolute wall clock time

The implemented cell-based gradient is exact for linear fields for any grid type whereas the edge-based approach is accurate only for certain mesh types. Furthermore, the performance of proposed cell-based gradient computation is better for most cell types; only for all-tet grids is it 1.15 slower than the edge-based method. In principle, the performance of the cell-based gradient computation scheme improves with decreasing ratio of cell count to node count (Table 2.8).

The enhancements come at the cost of the additional memory required for the cell-based approach, i.e. the coefficient matrix, the sum of element volumes

that coincide at each node, and the pointer from element to nodes list. The additional memory expressed as a percentage of the total solver memory used for the edge-based scheme is:

- 4.0% (JT-KIRK) / 22.2% (EXPLICIT) for tetrahedral grids
- 2.9% (JT-KIRK) / 12.2% (EXPLICIT) for hexahedral grids

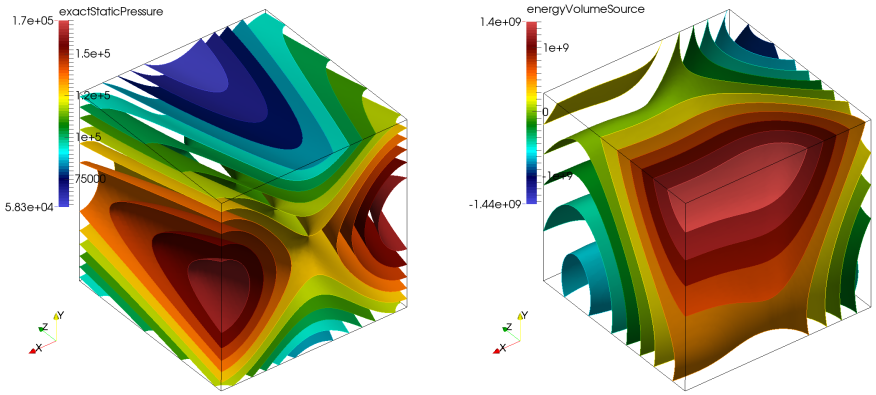
Less than 4% of memory increase for the default/key solver (JT-KIRK) used in STAMPS is a reasonable cost for the improved accuracy.

Summary

It was shown in this section that the proposed cell-based gradient computation method (see Section 2.3.3) is exact for linear fields for any mesh type. Although the cell-based method to calculate gradients is standard, a original approach was proposed for its derivation. In this approach a symbolic mathematical toolbox (SymPy) was used for derivation and reduction of number of floating point operations required for gradient computation (see Appendix A.4 for details). The computational cost of the cell-based approach is similar to the base edge-based technique. Hence, it is recommended that the accurate cell-based gradient computation should be used as a default in the STAMPS code.

2.7.3 Spatial discretisation accuracy

A method of manufactured (made-up) solution is used for verification of the STAMPS solver accuracy. The concept of manufactured solution is to define the solution vector $\vec{\vec{U}}$ using continuous functions of x, y, z-coordinates, and derive the appropriate source terms that arise from the imposed solution. An example pressure field and a corresponding energy equation source term is presented in Figure 2.25. A detailed description of the manufactured solution used for the STAMPS solver verification is shown in Appendix A.5. Only the accuracy of the Euler solver is investigated in this work.



(a) Pressure (p) (b) Energy source (f_e)

Figure 2.25: The 3D supersonic manufactured solution

The derived volume source terms that arise from the made-up solution were implemented in the code. With the sources imposed at each mesh node, and the exact solution set at the ghost boundaries as described in Section 2.3.10, the discrete solution should converge to the exact manufactured solution $\vec{\widetilde{U}}$ as the effective mesh size h_e converges to zero. Furthermore, a slope of convergence of a error norm with respect to the effective mesh size h_e provides information about the accuracy of the discretisation scheme. For a 2^{nd}-order accurate solver the norm of the error should reduce at a rate of h_e^2 (halving effective mesh size should reduce norm of the error by factor 4).

In order to investigate the accuracy of the discretisation scheme a sequence of uniformly refined grids is generated and the solution error norms are calculated for each grid level. The three different mesh types has already been introduced in Section 2.7.1.

The two norms, L_1 (Eq. 2.105), and L_∞ (Eq. 2.106) are considered in the analysis of convergence slope of discretisation (solution) errors (DE).

$$||\epsilon_h^\phi||_{1,N} = \left(\sum_{i=1}^{N} \left| \widetilde{\phi}_i - \phi_{h,i} \right| \right) \; / \; N \tag{2.105}$$

$$||\epsilon_h^\phi||_{\infty} = \max_i \left| \widetilde{\phi}_i - \phi_{h,i} \right| \tag{2.106}$$

The variable ϕ corresponds to a flow variable from the state vector $\vec{U}$, and N is the

dimension of the vector ϕ (number of mesh nodes). The tilde above the variable indicates an exact solution, e.g. $\widetilde{\rho}_i$ is an exact density variable evaluated at the discrete space h and mesh node i. Furthermore, the norms are calculated in three locations in order to investigate potential origins of reduced order of accuracy of the solver. For each location the characteristic mesh size is calculated:

- entire computational domain: $h_e = \left(\frac{1}{N}\right)^{1/3}$
- only interior nodes: $h_{ein} = \left(\frac{1}{N_{in}}\right)^{1/3}$
- only boundary nodes: $h_{eb} = \left(\frac{1}{N_b}\right)^{1/3}$.

In some cases, it is also useful to analyse a truncation error convergence slope in order to understand better the origins of reduced accuracy of the solver, if any. The truncation error TE_h is the difference between a mathematical model (PDE) and its discrete approximation, or in other words, it is the error due to the truncation of the continuous model. When the exact solution is known, the truncation error can be calculated exactly ($\widetilde{TE_h}$), see Eq. 2.107.

An exact solution $\vec{\widetilde{U}}$ is first evaluated at each grid node of the computational mesh and then the discrete residual is computed as usual (see Section 2.3). The error norms of $\widetilde{TE_h}$ can be calculated in the same way as for the discretisation error (DE) using Eqs. 2.105 and 2.106 with ϕ replaced with a residual of a given equation.

$$\widetilde{TE_h} = -\vec{R}_h\left(\vec{\widetilde{U}}^n_h, \bar{\mathcal{M}}_h\right) \Big/ \Omega_h \tag{2.107}$$

There are three main contributors determining the overall spatial discretisation accuracy of the residual $\vec{R}(\vec{W}$. To achieve a p^{th}-order accurate scheme the following conditions must be met.

1. **Solution reconstruction accuracy**. The solution vector $\vec{U}$ should be reconstructed exactly at the flux face for the $(p-1)^{th}$-degree polynomial. This requires gradient calculation accuracy to be $(p-1)^{th}$-order and the reconstruction point to be correctly selected, that is, be a flux-face Gauss point. In this work, the default cell-based gradient calculation accuracy was confirmed to meet the requirements - see Section 2.7.2. However, the integration point is taken as an edge midpoint (see Eq. 2.14), which is a

valid Gauss node for regular grids and general tet meshes; for the latter it is non-intuitive, but can be rigorously proved [130]. Hence, it is expected that the order of accuracy for a general (non-regular) hex, prism, pyramid or mixed type mesh will be reduced as compared to the $O(h_e^2)$ design order of accuracy.

2. **Flux integration accuracy**. Flux integration should be exact for fluxes represented as polynomials of $(p-1)^{th}$ degree. In this work, the flux integration accuracy is inspected through the analysis of the truncation error convergence evaluated using Eq. 2.107. Note that the accuracy of $\widetilde{TE}$ is an order of magnitude reduced as compared to the integration accuracy, as per its definition. Hence, a p^{th}-order accurate integration will result in $(p-1)^{th}$ truncation error convergence [131, 130, 24]. Flux integration accuracy assumes no errors originating from solution reconstruction (see point 1). Thus any errors in the solution reconstruction will affect the truncation error order of accuracy.

3. **Volume source term integration accuracy**. A volume integral should be exact for the source term function $\vec{q}$ represented by the $(p-2)^{th}$ degree polynomial.

For more details on the above refer to [130].

To make the discretisation accuracy analysis more transparent, only the error norms of solution error in density and truncation error of continuity equation are discussed. The author confirmed that norms for other variables/equations follow the same trend.

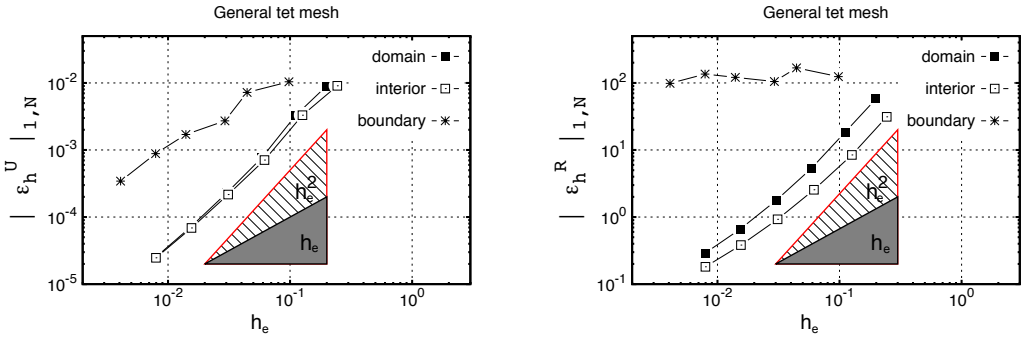
The default STAMPS solver settings are used for verification, that is, a ROE flux scheme and an accurate cell-based gradient (Section 2.3.5). No limiter is used.

Tetrahedral mesh

The discretisation scheme in STAMPS was expected to be 2^{nd}-order for the interior control volumes of a general tetrahedral grid; however, a slightly reduced

order can be seen in Figure 2.26a (interior). The reduced accuracy is most likely a result of a lack of consistent refinement in the grids sequence used for verification. The sequence of tetrahedral meshes was generated by defining an average edge size of a tet-element in ANSYS meshing platform. However, the condition is imposed in a weak manner and some discrepancies in edge sizes can arise, e.g. when the edge length of the cube divided by imposed element-edge-size is not a whole number. The influence of non-consistently refined grid sequence on accuracy testing is described in [71].

Figures 2.26a and 2.26b clearly show that the reduced order of accuracy comes from the boundaries where the integration point is not a Gauss point. Hence the boundary volume flux integration gives no reduction of truncation errors ($O(1)$) as the mesh is refined. This, in turn, leads to $O(h_e)$ discretisation errors at the boundaries.

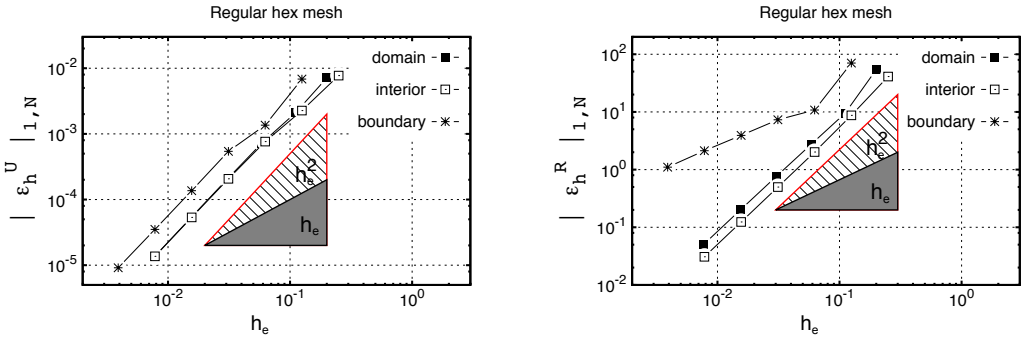


(a) Density discretisation error (DE) (b) Continuity truncation error (TE)

Figure 2.26: L_1 error norm convergence for a general tet mesh

Regular hexahedral mesh

The STAMPS solver is consistently second order accurate for regular hexahedral meshes (Figure 2.20b). This is confirmed by the convergence slope of the discretisation error norm shown on Figure 2.27a - $O(h_e^2)$.



(a) Density discretisation error (DE) (b) Continuity truncation error (TE)

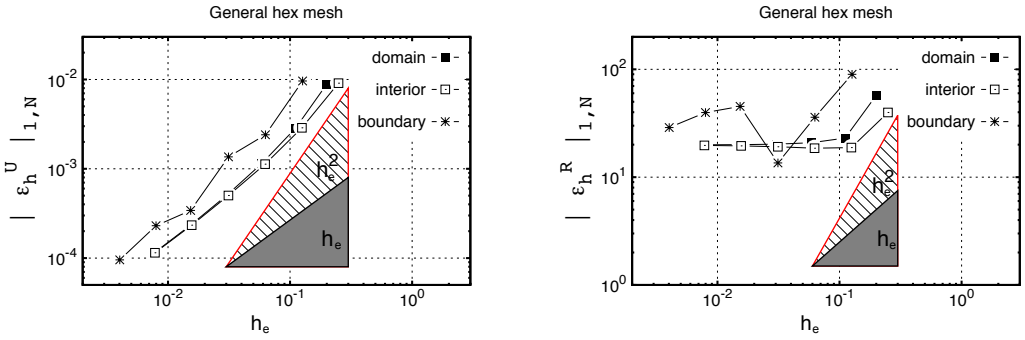
Figure 2.27: L_1 error norm convergence for a regular hex mesh

For this particular mesh type a supra-convergence is achieved for the truncation errors in the interior of the computational domain - Figure 2.27b. This is a well known behaviour for regular grids [63] and is a result of a perfectly symmetric stencil that leads to the cancellation of $O(h_e)$ errors in the domain interior. The truncation errors at the boundaries remain $O(h_e)$ as the stencil is no longer symmetric for the vertex-centred scheme. However, an order of magnitude accuracy reduction of the truncation error for the Euler system of equations does not affect the overall discretisation accuracy of the scheme [130], which can be confirmed when comparing boundary error norms for the DE (Figure 2.27a) and the $\widetilde{TE}$ (Figure 2.27b).

General (irregular) hexahedral mesh

For the irregular hexahedral grids (Figure 2.20c), a 1^{st}-order solution accuracy is obtained as confirmed by the slope analysis of converging error norms on Figure 2.28a. Furthermore, the truncation errors are $O(1)$ (Figure 2.28b), which indicates that the flux integration is first order accurate as expected for this grid type.

First order solver accuracy is also expected for general and mixed grid types where the elements are irregular.



(a) Density discretisation error (DE) (b) Continuity truncation error (TE)

Figure 2.28: L_1 error norm convergence for a general hex mesh

2.7.4 Summary

The verification of the STAMPS inviscid solver showed that the standard edge-based discretisation scheme is only second order accurate on regular and all tetrahedral grids. Furthermore, the scheme accuracy is not consistent over the entire computational domain, and an order-of-magnitude accuracy degeneration is reported at boundaries. A reduced order of accuracy was obtained for general and mixed grids. It is caused by the inaccurate integration scheme. In the standard edge-based scheme, an edge centroid is used as a flux face integration point which is not a Gauss node. Hence, the integration is not exact for a general grid types.

Chapter 3

Tangent solver, adjoint solver, and shape sensitivities

This chapter discusses gradient-based shape optimisation methods and efficient calculation of shape sensitivities. Background information is first provided in Sections 3.2–3.2. Next, the derivation of the discrete tangent-linear and adjoint solvers, as well as implementation details, are provided in Sections 3.3–3.5. The final sensitivity computation, where the tangent-linear/adjoint variables are the most computationally expensive components of $\frac{dL}{d\alpha}$ is described in Section 3.6. Various examples of flow and adjoint fields obtained using STAMPS solver are presented in Section 3.7. Additionally, an example of adjoint-based shape optimisation is performed using Onera M6 wing (Section 3.8).

3.1 Introduction

Gradient-based optimisation methods require a derivative of an objective function L (e.g. lift, drag) with respect to design variables α (e.g. mesh coordinates, CAD parameters). The derivative $\frac{dL}{d\alpha}$ that is often called shape sensitivity is used to drive the optimisation process towards an optimum. An essential part of successful gradient-based optimisation is an efficient and accurate gradient computation method. When the number of design variables is high, the most basic finite difference (**FD**) method is prohibitively expensive in terms of a computational effort, and the accuracy of estimated gradient is questionable [103]. As an alternative,

the dual solver can be introduced to address the drawbacks of finite difference. In particular, the tangent linearisation (**TL**) or its adjoint equivalent (**ADj**) can be used - see Section 3.2.

In typical engineering applications, the number of design variables is much larger than the number of objective functions ($N_\alpha >> N_L$), which is the primary reason for a large popularity of adjoint solvers in CFD optimisation. On the other hand, when the number of cost functions is greater than the number of design variables the tangent solver becomes a more efficient way of calculating derivatives.

Additionally, the adjoint solution can also be used to obtain a robust output-based adaptation sensor [29, 25].

There are two different techniques used for the development of tangent and adjoint solvers, this is, the continuous and discrete approaches [26, 132, 133]. In the continuous approach, the system of adjoint/tangent partial differential equations is first derived using Lagrange multipliers and the primal flow model (e.g. RANS equations). The obtained equations are then re-discretised and implemented in computer code. On the other hand, the discrete approach transforms the discretised flow equations directly. Algorithmic differentiation tools can be effectively used for this purpose, e.g. the source-transformation AD tool Tapenade [99], or Adol-C operator overloading tool [134]. The continuous approach allows the adjoint solver to be stabilised as the derived system is independently discretised from the primal solver, whereas the discrete approach allows an exact sensitivity to be achieved with respect to the output of the primal flow solver. Furthermore, the discrete adjoint/tangent-linear solver guarantees that the linear stability of the CFD code (primal) is inherited - see Figure 3.1 where the convergence slope of the primal and dual problem are the same at the linear convergence regime.

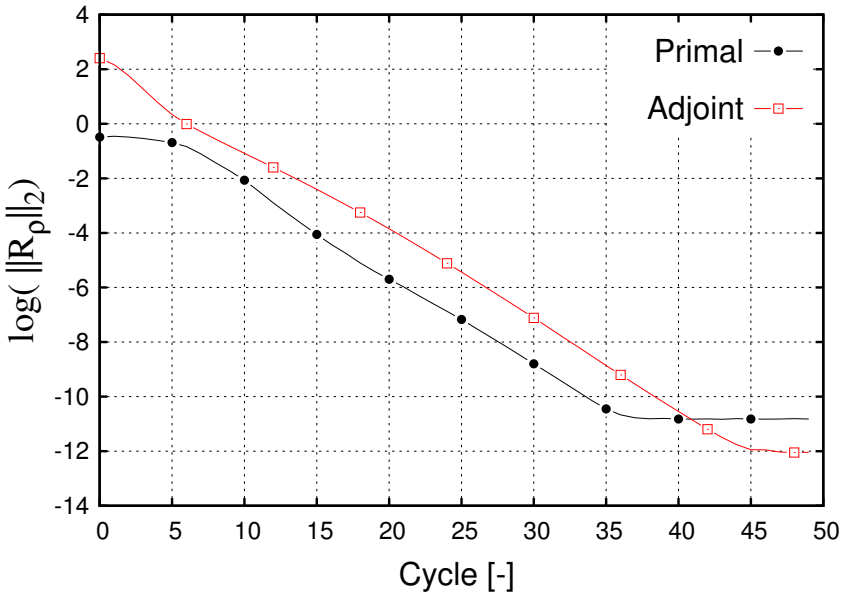


Figure 3.1: Equal convergence slope of flow and adjoint solver

In STAMPS, the derivatives are obtained using a source code transformation technique provided by the tool Tapenade [40]. This approach allows not only a fast differentiated code to be generated but also keeps the memory footprint for the adjoint solver at a similar level as for the primal flow solver, as opposed to the substantial memory requirements of the code generated by the operator overloading tools. Table 3.1 shows an example memory usage of the primal and adjoint solvers for the Onera M6 wing case.

Solver	Memory usage [GB]
Primal	3.229
Adjoint	4.006

Table 3.1: Memory requirement comparison of primal and adjoint solvers (JT-KIRK solver)

3.2 Adjoint equivalence

In most engineering applications the objective function L depends on the flow state W and design variables α (e.g. mesh coordinates). The sensitivity $\frac{dL}{d\alpha}$ is calculated using the chain rule as shown in Eq. 3.1. For transparency of equations the vector arrows are omitted in this section (e.g. for conservative state vector W).

The three partial derivatives can be distinguished. The first and last ($\frac{\partial L}{\partial W}$, $\frac{\partial L}{\partial \alpha}$) are usually low-cost and easy to calculate as they typically can be computed explicitly from the state W and design variables α and the differentiation of that relation is straightforward.

$$\frac{\partial}{\partial \alpha}\Big(L\Big(W(\alpha),\alpha\Big)\Big) = \frac{\partial L}{\partial \alpha} + \frac{\partial L}{\partial W}\frac{\partial W}{\partial \alpha} \tag{3.1}$$

The most computationally expensive partial derivative ($\frac{\partial W}{\partial \alpha}$) of the shape sensitivity ($\frac{\partial L}{\partial \alpha}$) is obtained from the linearisation of the flow equations, which can be written in short form as

$$R\big(W(\alpha),\alpha\big) = 0 \tag{3.2}$$

After applying the chain rule to Eq. 3.2 and rearranging it, the *tangent-linear system* is formed - Eq. 3.3. The short notation is used for each derivative, where $\widetilde{\mathbf{A}}$ stands for a Jacobian matrix of the governing equations (not to be confused with the approximate system matrix $\mathbf{A}$ used in Section 2.4), u is a tangent variable (not to be confused with the x-velocity vector component), and f is a right-hand side forcing term. In order to obtain complete matrix u, it is necessary to solve the tangent-linear system for each design variable α_i ($i = 1...N_\alpha$). The cost of a single tangent-linear system solve is similar to the cost of the original primal (flow) system solve, which makes this method not feasible for typical industrial cases, where the number of design variables usually exceeds thousands.

$$\frac{\partial R}{\partial W}\frac{\partial W}{\partial \alpha} = -\frac{\partial R}{\partial \alpha}, \quad \widetilde{A}u = f \tag{3.3}$$

As an alternative, the adjoint equivalence can be used to calculate the desired sensitivity with an affordable cost for typical industrial cases. The sensitivity (Eq. 3.1) can be re-written using the tangent solution obtained from the system solve Eq. 3.3 to give in Eq. 3.4.

$$\frac{dL}{d\alpha} = \frac{\partial L}{\partial \alpha} - \frac{\partial L}{\partial W}\frac{\partial R}{\partial W}^{-1}\frac{\partial R}{\partial \alpha} \tag{3.4}$$

Taking a transpose of Eq. 3.4 gives Eq.3.5.

$$\frac{dL}{d\alpha}^T = \frac{\partial L}{\partial \alpha}^T - \frac{\partial R}{\partial \alpha}^T \boxed{\frac{\partial R}{\partial W}^{-T} \frac{\partial L}{\partial W}^T} \tag{3.5}$$

The adjoint system of equations has a form presented in Eq. 3.6. For the single objective function L the adjoint variable ψ can be obtained with only a single system solve.

$$\frac{\partial R}{\partial W}^T \psi = \frac{\partial L}{\partial W}^T, \quad \widetilde{A}^T \psi = g \tag{3.6}$$

Rearranging Eq. 3.6 to get adjoint variable gives Eq. 3.7. A rectangular box is used to highlight identical terms appearing in Eq. 3.5 and Eq. 3.7.

$$\psi = \boxed{\frac{\partial R}{\partial W}^{-T} \frac{\partial L}{\partial W}^T} \tag{3.7}$$

Finally, the '*adjoint equivalence*' of Eq. 3.1 reads:

$$\frac{dL}{d\alpha} = \frac{\partial L}{\partial \alpha} - \frac{\partial R}{\partial \alpha}^T \psi \tag{3.8}$$

The cost of a single adjoint system solve is similar to the flow system solve. In this manner, the complete sensitivity vector can be obtained with a cost that is approximately twice the cost of the primal solution, and is independent of the number of design variables N_α. The memory required for adjoint calculation is around 24% higher than the memory required to run the flow solver as it was shown on the example M6 wing case in Section 3.2 in Table 3.1). The comparison was made for JT-KIRK solver.

In Sections 3.4 - 3.6, the details on tangent-linear and adjoint system implementation with the use of the source code transformation tool Tapenade [99] are described. Both systems are formed manually using selectively differentiated STAMPS subroutines described in Section 3.3. The same applies to the computation of final shape sensitivity $\frac{\partial L}{\partial \alpha}$. Note that in the following sections the mesh coordinates X will be used as design variables α. This semi-automatic approach allows run-time of the dual solvers to be reduced, and at the same time offloads the key differentiation effort to the Algorithmic Differentiation tool Tapenade.

3.3 Preparations of STAMPS routines for differentiation

The two key subroutines that are carefully prepared for differentiation with the Algorithmic Differentiation tool Tapenade are the `residual` and the `objective`. The subroutine heads are presented in the Pseudocodes 3.1-3.2. Both routines are organised such that their differentiation will result in the correct partial derivatives with respect to the state vector U without a need for manual operations. For example, in the case of `residual` subroutine (Pseudocode 3.1) the following functions: `bc`, `gradient`, and `limiter` could normally be calculated outside the `residual`. However, that would lead to an incorrect result for the $\frac{\partial R}{\partial U}$, as it would not include the influence of the boundary state, gradient, and limiter on the derivative.

```
1  subroutine residual(X↓, M↓, f↓, U_bc↓↑, U↓↑, R↑)
2
3    call bc           (X↓, M↓, U_bc↓↑, U↓↑)
4    call gradient     (X↓, M↓, U↓, ∇U↓↑)
5    call limiter      (X↓, M↓, U↓, ∇U↓, Φ↑)
6    call res_internal (X↓, M↓, U↓, ∇U↓, Φ↓, R↑)
7    call res_boundary (X↓, M↓, U↓, ∇U↓, Φ↓, R↑)
8    call source       (X↓, M↓, U↓, ∇U↓, f↓, R↑)
9    call res_bc       (M↓, R↑)
10
11 end subroutine residual
```

Pseudocode 3.1: Residual subroutine

The arrows next to the arguments in the routine signatures indicate whether the variable is an input ↓, an output ↑, or both ↓↑. The arguments of both functions are explained:

- `X`: vector of mesh coordinates. Note that the vector arrow is omitted to simplify notation for the purpose of this section.
- `M`: stands for mesh metrics introduced in Section 2.4, i.e. edge weights $\vec{s}_{ij}$, boundary weights $\vec{s}_{i_b}$, and volumes Ω.

- `f`: volume source term, Section 2.3.9.
- $\texttt{U}_{\texttt{bc}}$: ghost state vector, Section 2.3.10.
- `U`: flow solution vector. Note that the state `U` in the `residual` subroutine is an input and an output, $\downarrow\uparrow$. That is because, in general, the subroutine `bc` can modify the value of the state vector `U` at the boundaries when the hard boundary condition is imposed, Section 2.3.10.
- `L`: cost function value.
- Φ: limiter, Section 2.3.6.
- ∇`U`: gradient of flow variables, Section 2.3.3.
- `R`: residual vector.

In the `objective` subroutine 3.2, the calculation of gradient precedes other routine calls. It is required to take into account the influence of wall shear stress (viscous forces) on the derivative of aerodynamic forces (`lift`, `drag`, etc.) with respect to state vector `U`.

```
1  subroutine objective(X↓, U↓, L↑)
2
3    call gradient(X↓, M↓, U↓, ∇U↓↑)
4    select case(objective_type)
5      case(drag)       L = drag      (X↓, M↓, U↓, ∇U↓, F_drag↑)
6      case(lift)       L = lift      (X↓, M↓, U↓, ∇U↓, F_lift↑)
7      case(lift_drag)  L = lift_drag (X↓, M↓, U↓, ∇U↓, C↓_cnstr,
                                       F_cnstr↑)
8      case(ptloss)     L = ptloss    (X↓, M↓, U↓, PT_loss↑)
9      case(pdiff)      L = pdiff     (X↓, M↓, U↓, P↓_taget, P_diff↑)
10   end select
11
12 end subroutine objective
```

Pseudocode 3.2: Objective subroutine

To simplify the assembly of shape sensitivity and obtain partial derivatives $\frac{\partial L}{\partial X}$ and $\frac{\partial R}{\partial X}$ easily, the additional subroutine `RL` is created and differentiated with

Tapenade, Pseudocode 3.3. This subroutine gathers all calls to the functions that must be included in the differentiation process to obtain the correct shape sensitivity. For example, it includes the pre-processing step where the mesh metrics `M` and volume source terms `f` are calculated. This routine is created only for the purpose of sensitivity assembly and is not used anywhere else in STAMPS code as will be shown in Sections 3.4 and 3.5.

```
1 subroutine RL(X↓, U_init↓, U↓↑, R↑, L↑)
2
3   call metrics  (X↓, M↑)
4   call init     (U_init↓, U↑)
5   call source   (X↓, M↓, f↑)
6   call residual (X↓, M↓, f↓, U_bc↓↑, U↓, R↑)
7   call objective(X↓, M↓, L↑)
8
9 end subroutine RL
```

Pseudocode 3.3: RL subroutine

All the presented subroutines are defined with respect to the flow variables vector `U` to reflect the implementation details of STAMPS. However, the derivatives in Section 3.2 are using conservative variable vector `W` for derivations of sensitivity and dual systems. The inconsistency is addressed and treated appropriately in the following sections.

3.4 Tangent-linear solver

The tangent-linear system presented in Eq. 3.3 is rewritten as shown in Eq. 3.9 using the mesh coordinates vector `X` as the design variables α.

$$\frac{\partial R}{\partial W}\frac{\partial W}{\partial X} = -\frac{\partial R}{\partial X} \tag{3.9}$$

The tangent-linear system of equations is presented in terms of conservative state vector `W`. However, in STAMPS, the subroutines `residual` (Pseudocode 3.1) and `objective` (Pseudocode 3.2) are differentiated with respect to the state vector `U`. Using the variable transformation $\frac{\partial U}{\partial W}$ we can rewrite Eq. 3.9 as Eq. 3.10,

which forms the tangent-linear residual $\mathtt{R_{TAN}}$.

$$R_{TAN} = \left(\frac{\partial R}{\partial U}\frac{\partial U}{\partial W}\frac{\partial W}{\partial X} + \frac{\partial R}{\partial X}\right) = \left(\frac{\partial R}{\partial U}\frac{\partial U}{\partial X} + \frac{\partial R}{\partial X}\right) \tag{3.10}$$

A single iteration of the tangent-linear system solve for the column vector $u_{x,j} = \frac{\partial W}{\partial x_j}$ (x-coordinate of the design node `j`, i.e. $\mathtt{Xd(1,j)}$) is shown in Pseudocode 3.4.

```
1  Xd(:,:) = 0.0
2  Xd(1,j) = 1.0
3  ! Note: Rd ← ∂R/∂x_j
4  call RL_d_x(X↓, Xd↓, ..., R↑, Rd↑)
5
6  tangent-linear solver:
7
8    R_TAN = Rd
9    ! Note: u^n_{x,j} = ∂W^n/∂x_j
10   Wd = u^n_{x,j}
11   ! Note: Ud ← ∂U/∂W ∂W/∂x_j = ∂U/∂x_j
12   call flowVariables_d(W↓,Wd↓,U↑,Ud↑)
13
14   ! Note: Rd ← ∂R/∂U u_{x,j}
15   call residual_d_u(..., U↓, Ud↓, R↑, Rd↑)
16   R_TAN = R_TAN + Rd
17
18   δu^n_x = linearSolver(P, R_TAN)
19   u^{n+1}_x = u^n_x + δu^n_x
```

Pseudocode 3.4: A single iteration of tangent-linear solver

The partial derivative column vector $\mathtt{Rd} = \frac{\partial R}{\partial x_j}$ is obtained using the differentiated `RL` subroutine with respect to the node coordinates `X` using an appropriate seed vector $\mathtt{Xd(1,j)} = \mathtt{1.0}$ - see line 1-3 in the Pseudocode 3.4. The insertion '`_d`' in the function head indicates a differentiated routine, whereas the suffix '`d`' in the variable name indicates the derivative (output ↑) or a seed vector (input ↓). The suffices `_x` and `_u` correspond to the variable with respect to which the subroutine was differentiated. The dots in the arguments' lists replace variables which are not relevant in the given context to make the code more transparent. **P** is the ILU preconditioner based on the first-order Jacobian (for JT-KIRK solver),

which is identical to the primal solver preconditioner $\mathbf{P}$ presented in Section 2.4. The function call in line 18 of Pseudocode 3.4 stands for the linear system solve (GMRES used). The code structure of the tangent-linear solver is almost the same as the primal flow solver procedure (see Algorithm 1), with the main difference being that the differentiated residual function is used and the system has a non-zero right-hand side $\texttt{f}_\texttt{j} = \texttt{Rd} = \frac{\partial R}{\partial x_j}$, as in Eq. 3.3.

3.5 Adjoint solver

The adjoint system from Eq. 3.6 is rewritten as presented in Eq. 3.11 using the mesh coordinates vector `X` as the design variables α.

$$\frac{\partial R}{\partial W}^T \psi = \frac{\partial L}{\partial W}^T \tag{3.11}$$

The adjoint system of equations is presented in terms of conservative state vector `W`. However, in STAMPS, the `residual` subroutine (Pseudocode 3.1) and `objective` subroutine (Pseudocode 3.2) are differentiated with respect to the state vector `U`. Using the variable transformation $\frac{\partial U}{\partial W}^T$ we can rewrite Eq. 3.11 as Eq. 3.12, which forms the adjoint residual $\texttt{R}_\texttt{ADJ}$.

$$R_{ADJ} = \left(\frac{\partial R}{\partial W}^T \psi - \frac{\partial L}{\partial W}^T \right) = \frac{\partial U}{\partial W}^T \left(\frac{\partial R}{\partial U}^T \psi - \frac{\partial L}{\partial U}^T \right) \tag{3.12}$$

A single iteration of the adjoint system solve including adjoint residual accumulation and required transformations can be seen in Pseudocode 3.5.

```
1  Lb = 1.0
2  ! Note: Ub ← (∂L/∂U)^T
3  call objective_b_u(X↓, U↓, Ub↑, L↑, Lb↓)
4
5  solver_b:
6    R_ADJ = - Ub
7    Rb = ψ^n
8    ! Note: Ub ← (∂R/∂U)^T v
9    call residual_b_u(..., U↓, Ub↑, R↑, Rb↓)
10   R_ADJ = R_ADJ + Ub
11
12   Ub = R_ADJ
13   ! Note: R_ADJ ← (∂U/∂W)^T ((∂R/∂U)^T ψ − (∂L/∂U)^T)
14   call flowVariables_b(W↓,R_ADJ↑,U↑,Ub↓)
15
16   δψ^n = linearSolver(P^T, R_ADJ)
17   ψ^{n+1} = ψ^n + δψ^n
```

Pseudocode 3.5: A single iteration of adjoint solver

As the Tapenade reverse mode is used all the differentiated subroutines and variables gain the suffix 'b'. $\mathbf{P^T}$ is the transposed ILU preconditioner (JT-KIRK solver), which is the transposed equivalent of the primal solver preconditioner $\mathbf{P}$ presented in Section 2.4.

The pseudo-time stepping framework from the primal flow solver (see Algorithm 1) is also re-used for the adjoint solver.

3.6 Shape sensitivity accumulation

The tangent linear solution $u = \frac{\partial U}{\partial X}$ and the adjoint solution ψ are the most computationally expensive terms of the shape sensitivity $\frac{dL}{dX}$. The remaining terms of $\frac{dL}{dX}$ are accumulated in STAMPS using a semi-automatic approach. As mentioned in Section 3.3, to simplify sensitivity assembly all subroutines that require differentiation with respect to the mesh coordinates X were collected in the single routine `RL`. The sensitivity assembly using forward (TAN) and reverse (ADJ) differentiation modes of the Tapenade AD tool are described below. More details on the efficient usage of forward and reverse differentiated subroutines are

given in Appendix A.6.

TAN - forward differentiation mode

The final shape sensitivity for an example x-coordinate of node j, i.e. (x_j), can be accumulated using a tangent variable resulting from the tangent-linear system solve (see Section 3.4) and a forward differentiated objective function L with respect to mesh coordinates X and state vector U. The total derivative is presented in Eq. 3.13.

$$\frac{dL}{dx_j} = \frac{\partial L}{\partial x_j} + \frac{\partial L}{\partial W}\frac{\partial W}{\partial x_j} \tag{3.13}$$

As the subroutines in STAMPS are defined with respect to the flow variables vector U the appropriate transformation has to be applied as shown in Eq. 3.14

$$\frac{dL}{dx_j} = \frac{\partial L}{\partial x_j} + \frac{\partial L}{\partial U}\frac{\partial U}{\partial W}\frac{\partial W}{\partial x_j} \tag{3.14}$$

The pseudo-code for the (TAN) sensitivity assembly is as shown in Pseudocode 3.6.

```
1  Xd(1,j) = 1.0
2  ! Note: Ld ← ∂L/∂x_j
3  call RL_d_x(X↓, Xd↓, U_init↓, U↓, R↑, L↑, Ld↑)
4  DLDX(1,j) = Ld
5
6  Wd = u_{x,j}
7  ! Note: Ud ← ∂U/∂W  ∂W/∂x_j = ∂U/∂x_j
8  call flowVariables_d(W↓,Wd↓,U↑,Ud↑)
9
10 ! Note: Ld ← ∂L/∂U ∂U/∂x_j
11 call RL_d_u(X↓, U_init↓, U↓, Ud↓, R↑, L↑, Ld↑)
12 DLDX(1,j) = DLDX(1,j) + Ld
```

Pseudocode 3.6: Sensitivity assembly, TAN

Note that only a sensitivity for a single design point j and coordinate x is obtained - $\frac{dL}{dx_j}$. In order to accumulate the complete sensitivity vector for all design variables and coordinates (x, y, z), the tangent-linear system has to be solved $3 \times N$, where N is number of design nodes. However, in a practical application only the normal to boundary movement of nodes is considered to prevent problems with grid quality and satisfy smoothness requirements. Hence, in practical

applications the tangent-linear system has to be solved $1 \times N$ times.

ADJ - reverse differentiation mode

The total derivative $\frac{dL}{dX}$ can be obtained using reverse differentiated RL subroutine (Section 3.3) and the adjoint variable ψ resulting from the adjoint system solve - see Section 3.5. The reverse-mode computation of the shape sensitivity is obtained by transposing Eq. 3.4 as shown in Eq. 3.15.

$$\frac{dL}{dX}^T = \frac{\partial L}{\partial X}^T - \frac{\partial R}{\partial X}^T \psi \tag{3.15}$$

The pseudo-code for the (ADJ) sensitivity assembly is shown in Pseudocode 3.7.

```
1 Lb = 1.0
2 Rb = -ψ
3 call RL_b_x(X↓, Xb↑, U_init↓, U↓, R↑, Rb↓, L↑, Lb↓)
4 ! Note: Xb ← ∂L/∂X^T − ∂R/∂X^T ψ
5 DLDX = Xb
```

Pseudocode 3.7: Sensitivity assembly, ADJ

The complete derivative for all $(3 \times N)$ design variables can be calculated using only a single call to the reverse differentiated subroutine `RL`. The variable (seed) `Lb=1` is used to obtain the partial derivative $\frac{\partial L}{\partial X}^T$, and the adjoint variable seed vector `Rb` $= -\psi$ is used to get matrix-vector product $-\frac{\partial R}{\partial X}^T \psi$. As a result, the complete derivative $\frac{dL}{dX}$ is accumulated in the output variable `Xb`.

Verification

In STAMPS, the initial verification of the sensitivities obtained using adjoint solver and reverse differentiated subroutines was performed by Christakopolous [129]. A comparison of sensitivities between adjoint, tangent, and finite difference was carried out showing good match between the three approaches.

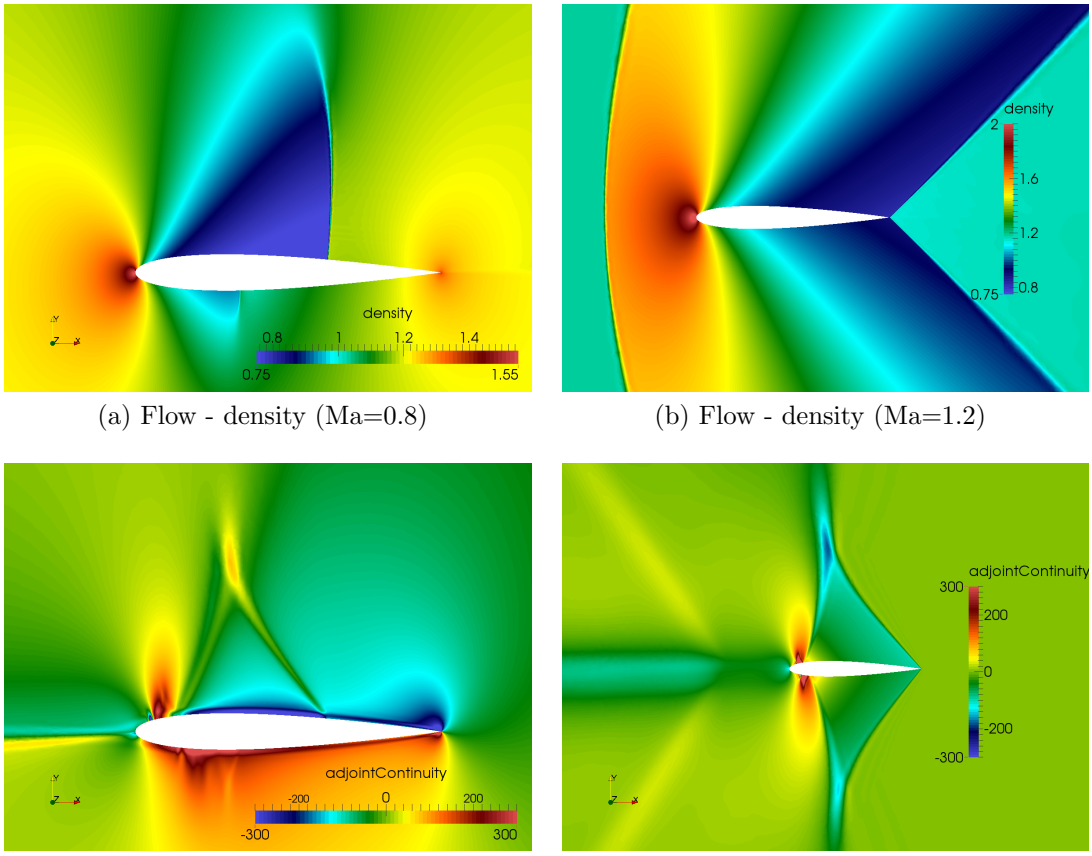
A more rigorous verification using convergence of a Taylor series reminder was proposed by Farrell et al. [76]. An alternative approach based on the known exact solution was presented by Nemec [75]. Both approaches are discussed Chapter 6 and they are recommended for the future verification work of adjoint solver in STAMPS.

3.7 Flow and adjoint examples

A range of cases is used in this section to show examples of flow and adjoint solutions as well as shape sensitivity fields.

NACA0012

Figure 3.2 shows a simple 2D inviscid NACA0012 case for two flow conditions. In the left column (Figures 3.2a and 3.2c), the transonic flow results ($Ma_\infty = 0.8$) are shown, and in the right column (Figures 3.2b and 3.2d), the supersonic flow results ($Ma_\infty = 1.2$). In both cases angle of attack is set to $AoA_\infty = 1.25°$, and the cost function is defined as the drag force. In the case of transonic flow, there is a strong shock formed at the upper surface of the aerofoil and a weak shock at the bottom, whereas for the supersonic case there is a bow shock and two oblique shocks originating from the trailing-edge of the aerofoil. Example adjoint continuity fields (Figure 3.2c and 3.2d) show how the insertion of a mass source would affect the cost function of interest (drag). A negative mas source (i.e. mass sink) imposed at the lower aerofoil section would decrease the drag as the adjoint field is positive there. For the supersonic flow conditions (Figure 3.2d), we can clearly see that the adjoint variable is zero behind the aerofoil trailing edge. This is expected as the flow conditions are supersonic ($V > c$) and no disturbance can propagate faster than the speed of sound. Therefore the zero $'source–sensitivity'$ indicates that the addition of any mass source behind the aerofoil trailing edge will have no influence on the drag. That is in contrast to the subsonic case where the high sensitivities are seen around the trailing-edge, Figure 3.2c.



(a) Flow - density (Ma=0.8)

(b) Flow - density (Ma=1.2)

(c) Adjoint - continuity (Ma=0.8)

(d) Adjoint - continuity (Ma=1.2)

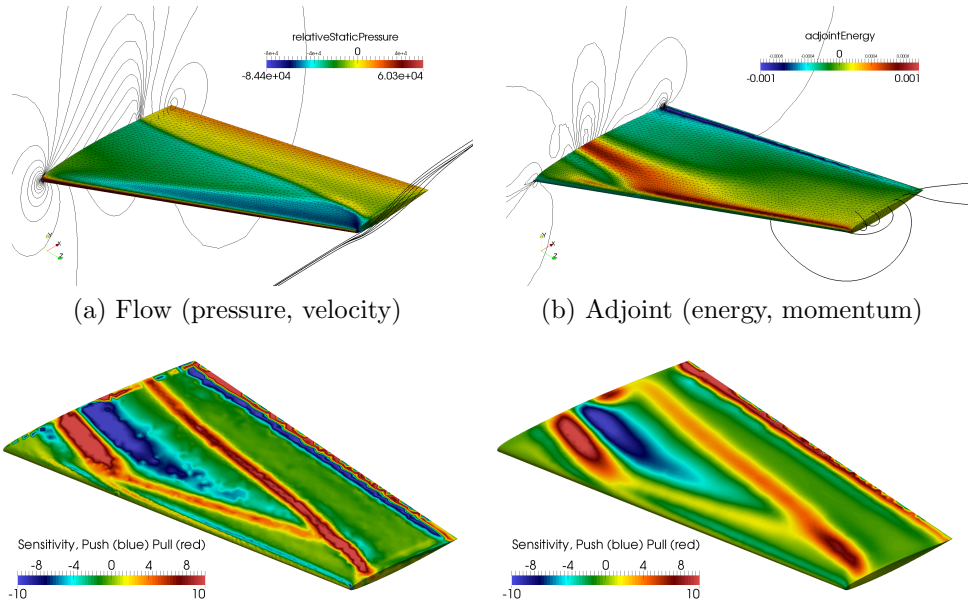
Figure 3.2: Naca0012 flow and adjoint results. Ma=0.8 - left column, Ma=1.2 - right column

Onera M6 wing

Results of a 3D transonic Onera M6 wing case [106] are shown in Figure 3.3. The flow conditions are $Ma_\infty = 0.8395$, $AoA_\infty = 3.06°$, and the objective function is defined as a drag with lift constraint applied as a penalty function as shown in Eq. 3.16. The value of the constant for the lift constraint can be adjusted to either increase or decrease importance of the lift force change on the objective L. In this particular case the value 4 was chosen (higher than 1) to increase the penalty on decreasing lift. Other case information can be found in Section 4.3. The goal of the optimisation would be to reduce drag while keeping the lift unchanged. Example flow and adjoint contours, as well as streamlines are presented in Figures 3.3a and 3.3b. Figure 3.3c shows surface shape sensitivity on the wing surface without any post-processing, whereas Figure 3.3d presents sensitivity after five explicit Laplacian smoothing iterations to remove unwanted

high-frequency modes that would lead to a very noisy and impractical shapes. The explicit Laplacian smoothing is described in more detail in Appendix A.7. Constant $\beta = 1$ was used.

$$L = F_D + 4\left(F_L - F_L^0\right)^2 \tag{3.16}$$



(a) Flow (pressure, velocity)

(b) Adjoint (energy, momentum)

(c) Normal sensitivity

(d) Normal sensitivity after smoothing

Figure 3.3: Inviscid M6 wing - flow, adjoint, sensitivity

DrivAer

An example of a large industrial level application of STAMPS flow and adjoint solvers is shown in Figures 3.4–3.7. The key case information is as follows.

- Case: DrivAer car[24] [135]. Half of the car used for the simulation with the symmetry plane.
- Mesh type/size: tet + prism in the boundary layer / 1.39 million nodes, 5.72 million cells.
- Flow: subsonic, turbulent (SA) with wall function ($y+ > 30$).
- Flow conditions: Ma = 0.113 (V = 140 km/h), Angle of attack $AoA = 0^\circ$, $T_\infty = 293.15$ K, $p_\infty = 101325$ Pa.

[24] http://aboutflow.sems.qmul.ac.uk/events/munich2016/benchmark/testcase4/

- Objective function: drag.
- Solver setup: default options described in 2.1, i.e., 2^{nd}-order discretisation accuracy (flow/adjoint), ROE flux scheme, JT-KIRK solver, VEM limiter, Cell-based gradient.

Figure 3.4 shows the velocity magnitude contours on the symmetry plane and the static pressure contours elsewhere.

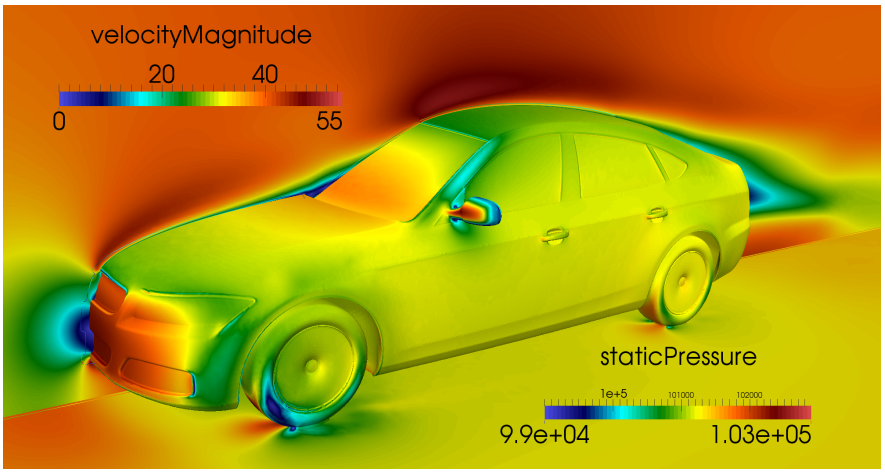
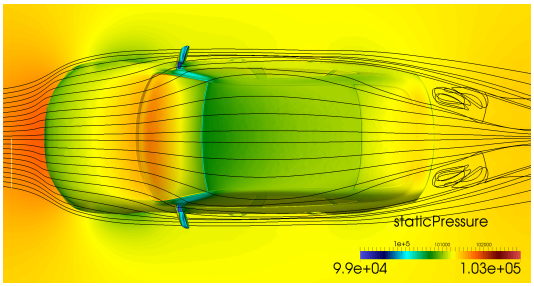
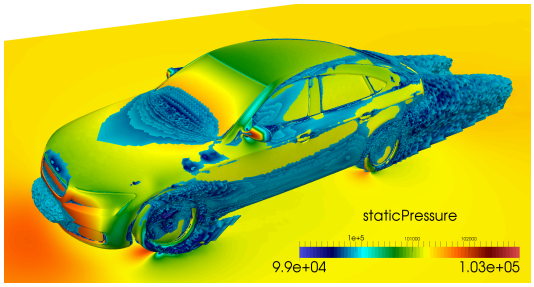


Figure 3.4: DrivAer case flow solution

A large flow re-circulation area is present behind the notchback-type car body as indicated in Figures 3.5a - velocity streamlines, and 3.5b - iso-volumes of velocity between $2 - 15$ m/s ($7.2 - 54$ km/h). Very strong separations are also visible around the wheels, extending along the bottom of car doors. The stagnation areas are visible at the front bumper and near the bottom line of the windscreen. The car body was mirrored for visualisation purposes in the post-processing tool Paraview.



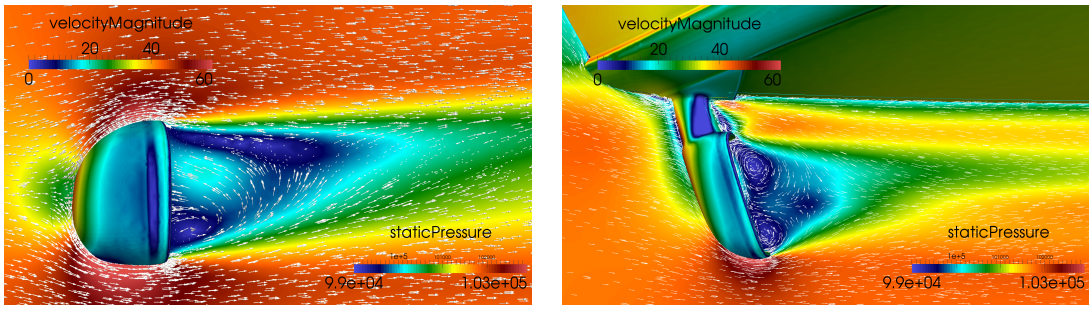
(a) Streamlines



(b) Low velocity iso-volumes

Figure 3.5: Separation and recalculation regions for drivAer case

Details of the flow around the car mirror are highlighted in Figure 3.6. Velocity vectors that shows re-circulation areas behind the side-mirror and velocity magnitude contours are shown in section planes, whereas static pressure contours are displayed on the car body.



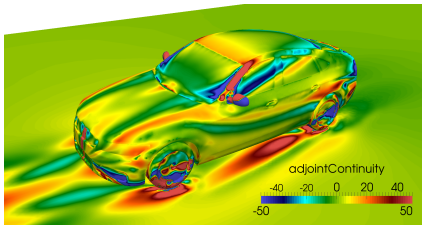
(a) Side view (b) Top view

Figure 3.6: Flow around DrivAer mirror

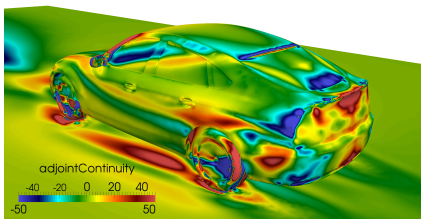
Figure 3.7 gathers all adjoint field contours; left column - front iso-views, right column - back iso-views.

When analysing e.g. the y-momentum adjoint field (Figure 3.7e–3.7f) one can see large positive values in the area behind the mirror along the A-pillar and roof edge (front-side window). This indicates that injecting negative y-momentum would decrease the drag, where the positive y-direction is defined along the normal to the ground/road. The positive values of adjoint y-momentum in these areas could be justified by the fact that injection of negative y-momentum source would prevent the air from the car side to turn towards the car roof - see the low velocity 'patch' on Figure 3.5b that originates from the mirror and extends towards the back window. As a result, a reduction of the induced drag generated near the A-pillar is expected.

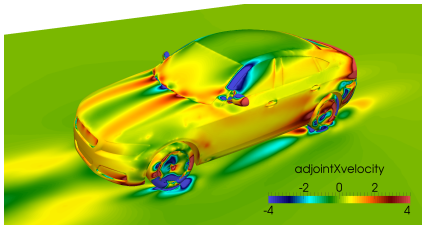
Similarly, the high x-momentum injected in the direction of the car motion (opposite to x-axis) would create high-pressure areas at the back of the car resulting in the generation of thrust force.



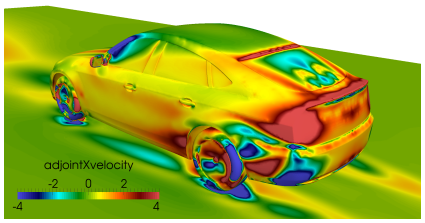
(a) Continuity



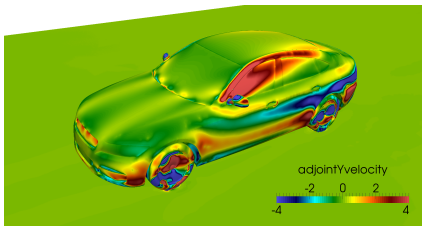
(b) Continuity back view



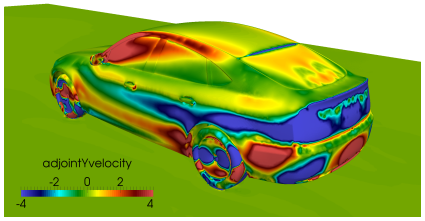
(c) X-momentum



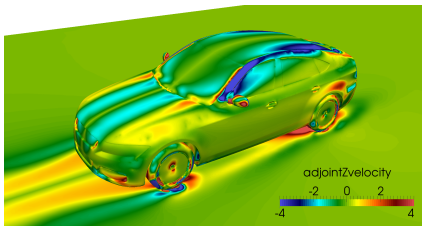
(d) X-momentum back view



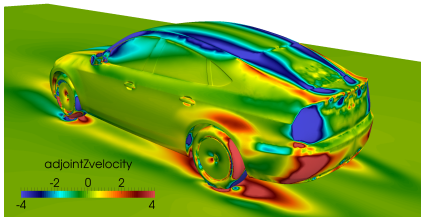
(e) Y-momentum



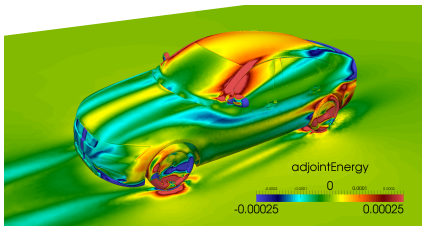
(f) Y-momentum back view



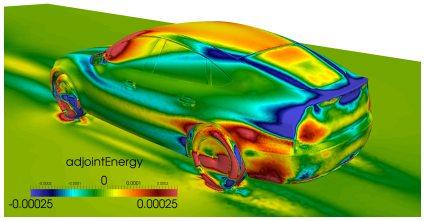
(g) Z-momentum



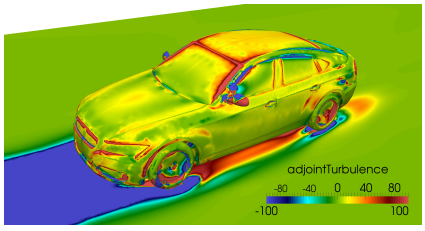
(h) Z-momentum back view



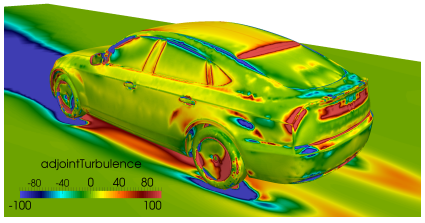
(i) Energy



(j) Energy back view



(k) Turbulence



(l) Turbulence back view

Figure 3.7: DrivAer adjoint solution

3.8 Optimisation example

STAMPS has most components required for adjoint-based shape optimisation. However, optimisation is not a built-in functionality of STAMPS. To perform shape optimisation a script is required that executes the code in appropriate modes and order, and provides inputs to the optimiser of choice. The optimisation algorithm is presented in the form of simple flowchart in Figure 3.8.

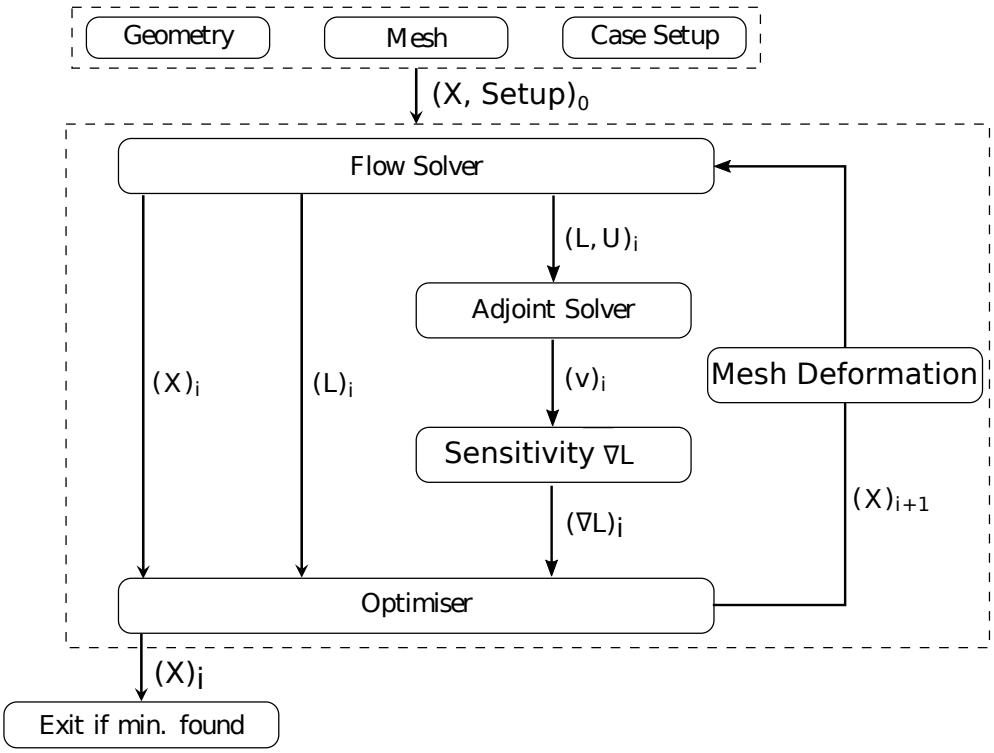


Figure 3.8: Optimisation flowchart

Onera M6 wing

A python script was created to show an example application of adjoint-based shape optimisation. A transonic Onera M6 wing case was used at Ma = 0.84 and angle of attack $AoA = 3.06°$. Mesh nodes (x-, y-, z-coordinates) that describe wing geometry were used as design variables. The objective of optimisation was defined as drag coefficient (C_D) with a penalty on lift coefficient (C_L), where a custom constant (4) was chosen to control strength of the lift penalty function (see Eq. 3.17). The variable C_{L0} is the initial lift coefficient value of the undeformed M6 wing. The deformation of the root aerofoil of the wing was fixed in z-direction

to maintain the symmetry plane.

$$L = C_D + 4(C_L - C_{L0})^2 \tag{3.17}$$

A simple line-search optimisation algorithm with Armijo rule [146] was used as an optimiser. In total 24 optimisation iterations were performed and objective function was reduced by 31%. Figure 3.9 shows convergence history of the objective function.

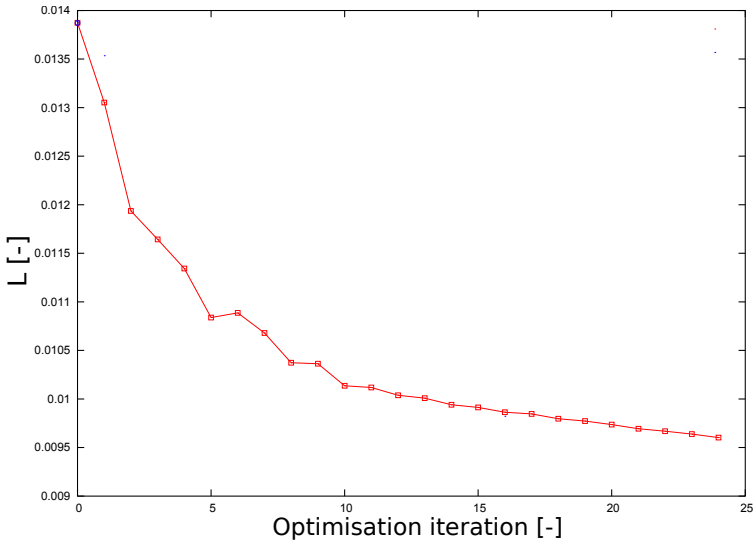
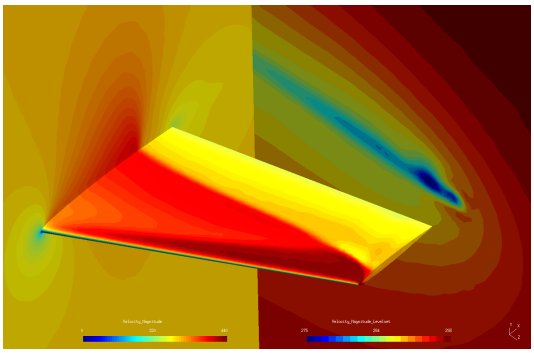
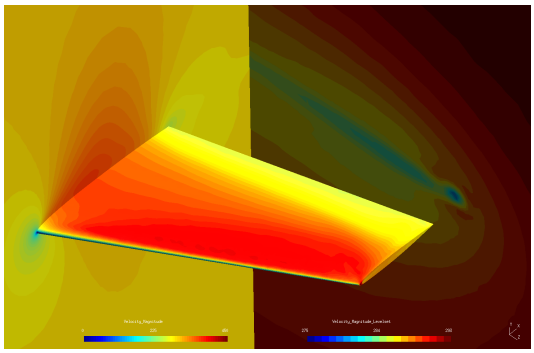


Figure 3.9: Convergence history of objective function

Velocity and pressure contour plots are shown in Figures 3.10 and 3.11. It is clear the optimisation removed shocks present on the initial Onera M6 wing and reduced wake behind the wing (Figures 3.10) which led to reduction of the drag coefficient while maintaining the initial lift coefficient.

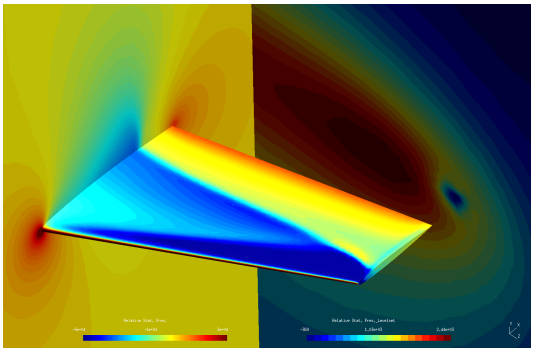


(a) Initial M6 wing

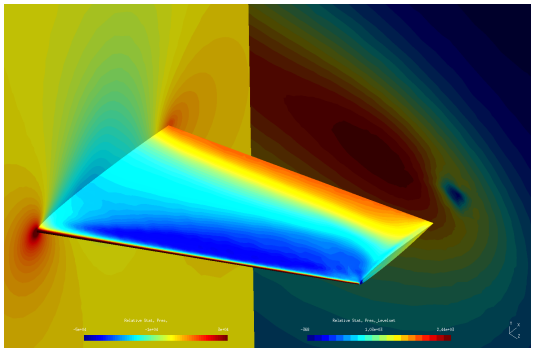


(b) M6 wing after 24 optimisation steps

Figure 3.10: Velocity contours comparison



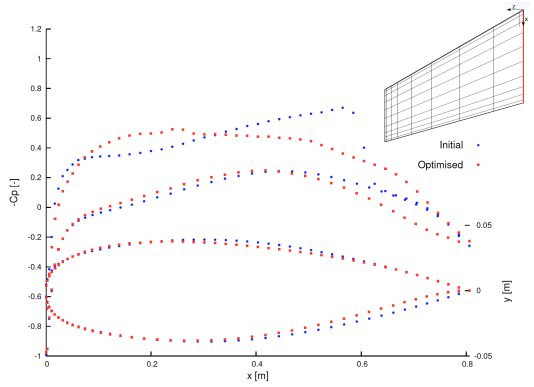
(a) Initial M6 wing



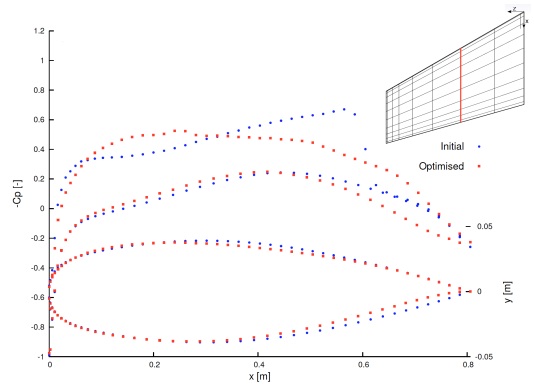
(b) M6 wing after 24 optimisation steps

Figure 3.11: Pressure contours comparison

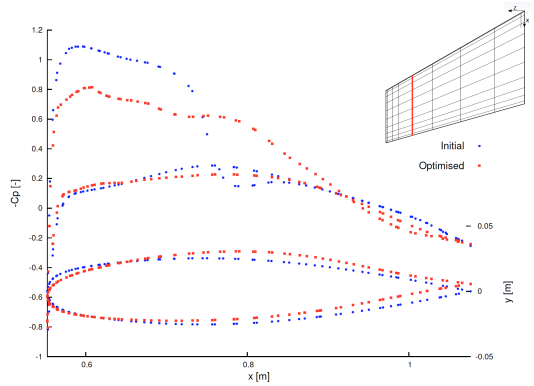
Figure 3.12 shows a comparison of wing profile shape and pressure coefficient distribution between the initial and final shape. The results are presented in 4 wing sections.



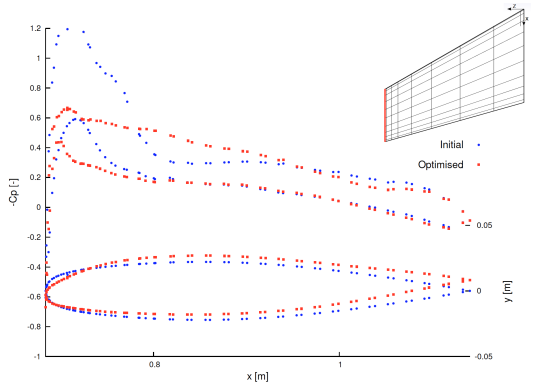
(a) Section 1



(b) Section 2



(c) Section 3



(d) Section 4

Figure 3.12: Shape change and pressure coefficient distribution in 4 sections - comparison between initial and improved M6 wing

Chapter 4

Solver enhancements

4.1 Introduction

Although the adjoint-based methodology made shape optimisation affordable, the typical cost of the process is still at least an order of magnitude higher than obtaining a flow solution only. One-shot optimisation is an example method that can lead to time savings [136]. It is based on driving the objective function to the optimum using a minimal acceptable accuracy of the flow and adjoint solutions instead of fully converging those results at each optimisation cycle. In other words, the flow and adjoint equations are solved progressively, that is, the convergence level is decreasing with evolving shape of the geometry that is being optimised. Although one-shot approach allow to reduce computational cost of optimisation by reducing total number of solver iterations it is still at least 5-10 times the cost of primal evaluation, and that is under the assumption of optimally tuned parameters. Hence, looking for further time savings is still an important aspect of successful applications of shape optimisation. In this chapter, methodologies that allow reductions in the total run-time of the flow solver to be achieved are presented.

Implicit solvers allow for large time steps and hence can lead to a faster convergence compared to explicit schemes - see Section 2.4. However, for the nonlinear system of equations (e.g. RANS), implicit solvers often suffer from poor initialisation and the requirement for a small initial time step. When a physical

problem is highly nonlinear and the initial state is far from the stationary point, the reduced time steps have to be used to prevent a solver from diverging. The step size is controlled through the CFL (Courant-Friedrichs-Lewy) number, where initially it is set to low values and can be increased as the solution progresses towards the stationary point. The adjustment of the CFL number is usually left to the user, and it is often a tedious trial and error process (case/application dependent). Hence, an automatic CFL adjustment method could be beneficial for solver robustness and run-time reductions.

Once the highly nonlinear convergence stage is passed and the stationary point is getting closer, an opportunity for further run-time reductions appears, i.e., turning off the ILU preconditioner re-computations (see Section 2.4). Although the computationally most demanding part of the implicit scheme is the linear system solve (subroutine `linearSolver` in Algorithm 1), the computation of the ILU preconditioner is also an expensive process. The preconditioner, which is based on the Jacobian of the 1^{st}-order discrete system of equations, is re-computed after each nonlinear iteration (pseudo time step). However, as the nonlinear solver progresses towards the stationary point the linear convergence regime is approached. In such a case, the Jacobian is no longer varying between the nonlinear steps and can be frozen. Hence, the computational effort can be reduced and the solution achieved faster.

In this chapter, an algorithm is presented that automatically adjusts the CFL number, and controls Jacobian re-computation - Section 4.2. It allows a robust implicit solver with a minimum user input to be achieved. Additionally, the following convergence acceleration techniques are presented in Sections 4.5 and 4.6 to further reduce solver run-time.

- Multigrid start-up (MGS) - allows a better initialisation state for the solver to be achieved by reaching a certain level of convergence on the coarse grids and transferring the solution to fine mesh.

- Low Mach scaling (LMS) - allows convergence for low Mach number flows to be enhanced by adjusting ambient conditions (p, T) to increase Mach number while keeping the Reynolds number unchanged.

- Residual-based time-stepping (RBTS) - this can be seen as a form of implicit local time-stepping. It recognises control volumes where the local time step can be increased and where it should be reduced based on the local residual norm.

- Enhancement for highly stretched meshes (AR) - this is another example of implicit local time-stepping where the cell aspect ratio (AR) is used to scale the time step value at each control volume. The time step is increased for high aspect-ratio cells to prevent slow convergence.

The methodologies are described, and the enhancements tested using two turbulent flow cases and one inviscid:

- 2D Rae2822 aerofoil

- 3D M6 wing

- 3D U-bend channel flow.

Details on the test cases are presented in Section 4.3

4.2 Stabilisation of implicit solver

This section describes main building blocks of a robust implicit solver implemented in STAMPS. The aim is to stabilise JT-KIRK solver against poor initial guess, and enhance its the overall performance. The CFL number and pseudo time marching scheme has already been introduced in Section 2.4. First, the overview-algorithm is presented that explains main additions to baseline JT-KIRK solver such as automatic CFL adjustment, Jacobian re-computation control, or safeguarding for a divergence of the solver. Next, each main building block is described in more detail and summarised using an algorithm.

4.2.1 Robust implicit solver

The procedure that shows main additions to the baseline JT-KIRK solver to increase its robustness is presented in Algorithm 3. The baseline STAMPS's solver, which has already been introduced in Algorithm 1, is enclosed in a single

subroutine call shown in line 1 of the robust solver algorithm. It represents a single update step of the solution. All the remaining parts of the procedure are related enhancements implemented by the author, and are executed at the end of each solver update.

Algorithm 3 Robust Solver

1: $\vec{U}^{n+1} \leftarrow \texttt{solver}\left(\vec{U}^{n}, \bar{\mathcal{M}}, CFL^{n}, (\vec{Q}_{MG})\right)$

2:

3: $NaN^{U}_{found} \quad \leftarrow \texttt{nancheck}(\vec{U}^{n+1})$

4: **if** (NaN^{U}_{found}) **then**

5: $\quad U^{n+1} \quad \leftarrow \vec{U}_{restart}$

6: $\quad CFL^{n+1} \quad \leftarrow 0.5\ CFL^{n}$

7: **else**

8: $\quad CFL^{n+1} \quad \leftarrow \texttt{cfl\_adjustment}\left(\vec{U}^{n+1}, \delta W^{n}, CFL^{n}, \vec{R}^{n}\right)$

9: $\quad \vec{U}_{restart} \quad \leftarrow \texttt{setrestartstate}\left(\vec{U}^{n+1},\ ||\vec{R}||_2^{min},\ ||\vec{R}^{n}||_2\right)$

10: $\quad \mathbf{A}_{recompute} \quad \leftarrow \texttt{acontrol}\left(N_{PWS},\ ||\vec{R}||_2^{min},\ ||\vec{R}^{n}||_2\right)$

11: **end if**

First, the solution at new pseudo time is calculated. Next, a $Not-a-Number$ (NaN) check is performed to make sure that the new solution represents valid floating point numbers (line 3). An L_2 norm of the state vector $\vec{U}^{n+1}$ is calculated for this purpose. If the condition fails, the solution is re-initialised and CFL number reduced. The initial value of $\vec{U}_{restart}$ is provided in the initialisation step executed once before the solver starts. The initial CFD number is provided by the user. If the NAN-check is successful, a set of functions is executed. The CFL adjustment and Jacobian re-computation control (`acontrol`) subroutines are described in Sections 4.2.2 and 4.2.3 as they require broader explanation.

The remaining `setrestartstate` routine (line 9) is used to assign a new restart state. As mentioned previously, the initial value of restart vector is set to initial flow conditions (freestream value in STAMPS). However, as the solution progresses it is beneficial to overwrite the restart state vector because very often the instability in the solver appears only after a certain number of solver iterations. Hence, to avoid loosing partially solved flow by using restarting with

an initial guess, a previous best restart condition is assigned and CFL number reduced to re-try the solver with a smaller update step. However, assignment of a new restart state has to be assessed carefully. In STAMPS, $\vec{U}_{restart}$ is updated only when the residual norm of each corresponding equation falls below the currently stored minimum. The lowest residual norm is then overwritten - $||\vec{R}||_2^{min} \leftarrow ||\vec{R}^n||_2$.

The computational cost associated with routine calls in lines 3, 9, and 10 in the Algorithm 3 is negligible, whereas the CFL-adjustment (line 8) is more time consuming. However, the results presented in Section 4.4.2 show that the benefit of automatic CFL-adjustment outbalances added computational effort and most importantly, it allows a robust solver with a minimal user interaction to be achieved.

4.2.2 Automatic CFL adjustment

Eq. 4.1 shows an implicit time integration scheme introduced in Section 2.4, where the system matrix **A** is defined in Eq. 4.2.

$$\left[\frac{\Omega_i}{CFL^n\,\Delta t_i^n} + \left(\frac{\partial \vec{R}}{\partial \vec{W}}\right)_i^n\right] \Delta \vec{W}_i^n = -\vec{R}_i^n \tag{4.1}$$

$$\mathbf{A}_i^n = \left[\frac{\Omega_i}{CFL^n\,\Delta t_i^n} + \left(\frac{\partial \vec{R}}{\partial \vec{W}}\right)_i^n\right] \tag{4.2}$$

When a good initial guess is available for solving system of flow equations the CFL number can be set to a very large value (CFL $\rightarrow \infty$), or in other words the first term in the system matrix shown in Eq. 4.2 can be set to 0. In this case, the system matrix **A** becomes equivalent to Jacobian of system of flow equations ($A \overset{CFL\rightarrow\infty}{=} \frac{\partial R}{\partial \vec{W}}$), and a Newton method is achieved that allows rapid convergence of the solution to the stationary point. However, in practice a good initial guess that guarantees stability of Newton solver is not known and a limited value of the initial CFL number has to be used. This allows to stabilise the solver by slowing down the evolving solution using a finite size time step. As the solution progresses the CFL number can be increased and Newton solver achieved.

In practical engineering applications, this is usually achieved by first running a number of iterations using low CFL number, and then restarting the solver with a higher value. This approach is very manual and far from optimal. Hence, an automatic CFL adjustment method is desired.

Adjustment of the CFL number can be automated using a common optimisation technique, i.e. a line search algorithm [103], where the objective function to be minimised is defined as a squared L_2-norm of residual (Eq. 4.3).

$$z^n = ||\vec{R}(\vec{U}^n)||_2^2 \tag{4.3}$$

The decision on increasing or decreasing the CFL number for next solver iteration is made based on the second Wolfe condition 4.4, where the constant η_1 is set to $1e-4$ - recommended by Nocedal and Wright [67]. Scale factor s is initialised with value of 1.

$$z^{n+1} - z^n \leq \eta_1 \; s \; \nabla z^n \; p^n \tag{4.4}$$

Michalak and Ollivier-Gooch presented a ramping technique that uses line-search to evolve CFL number as the solution progress and applied it to a compressible Euler solver [20]. In this work, the adjustment of CFL number was implemented for Euler and RANS solvers. Furthermore, the gradient of the objective function required in the line-search algorithm was calculated using an exact Jacobian of the discrete system of flow equations as opposed to the approximate Jacobian matrix used by Michalak and Ollivier-Gooch.

A detailed procedure implemented in STAMPS is presented in Algorithm 4. The CFL adjustment algorithm is executed after each cycle of the solver presented in (see Algorithm 1). Constants and user defined parameters are as follows

- $s^{up} = 1.1$ (constant), if the second Wolf condition in Algorithm 4 is satisfied the CFL number is increased by 1.1.
- $s^{down} = 0.5$ (constant), if the second Wolf condition in Algorithm 4 is violated or when $Not - a - Number$ (NaN) is found, the CFL number is

sharply decreased by 0.5.

- $CFL^0 \leftarrow CFL_{init}$ (user defined), starting value of CFL.
- $\vec{U}_{restart}$ restart state vector introduced already in Section 4.2.1, which is used in case of a failed update (NaN-check).

Algorithm 4 CFL-Adjustment

1:
2: `cfl_adjustment` $\left(\vec{U}^{n+1},\ \delta\vec{W}^{n},\ CFL^{n},\ \vec{R}(\vec{U}^{n})\right)$
3: $s \leftarrow 1.0$
4: $\vec{R}(\vec{U}^{n+1})$
5: $\triangle z^{n} \leftarrow z^{n+1} - z^{n}$
6: **if** (`isnan`$(\triangle z)$) **then**
7: $CFL^{n+1} \leftarrow s^{down}\ CFL^{n}$
8: $\vec{U}^{n+1} \leftarrow \vec{U}_{restart}$
9: **else**
10: $\nabla z^{n}\ \delta\vec{W}^{n} = \left(\vec{R}^{n}\right)^{T}\left(\left(\frac{\partial\vec{R}}{\partial\vec{W}}\right)^{n}\delta\vec{W}^{n}\right)$
11: **if** $\left(\ \triangle z^{n} \leq \eta_1\ s\ \nabla z^{n}\ \delta\vec{W}^{n}\ \right)$ **then**
12: $s \leftarrow s^{up}$
13: **else**
14: $s \leftarrow$ `max` $\left(s^{down},\ min\left(0.5\ s/\left(1 - \frac{\triangle z}{s\ RHS}\right),\ s^{up}\right)\right)$
15: **end if**
16: $CFL^{n+1} \leftarrow s\ \frac{||\vec{R}^{n}||_2}{||\vec{R}^{n+1}||_2}\ CFL^{n}$
17: **end if**
18:

First, the residual for an updated solution is calculated and the change in the objective $\triangle z$ is calculated (line 4 in Algorithm 4). Next, a $Not - a - Number$ (NaN) condition is assessed to check if the new residual value for the updated solution is a valid floating point number, or else the CFL is sharply reduced for the next solver iteration, and the state vector is re-initialised. If NaN check is successful, the terms required to assess second Wolfe condition are evaluated.

The gradient of the objective function (∇z) is calculated by the algorithmic differentiation tool Tapenade [99] applied to the `residual` subroutine introduced in Section 3.3 in Pseudocode 3.1. The Jacobian matrix $\frac{\partial \vec{R}}{\partial \vec{W}}$ is not stored explicitly in the memory; only the matrix-vector product is required. Consequently, the $\nabla z^n \ \delta \vec{W}^n$ term in line 10 of Algorithm 4 can be efficiently calculated using forward differentiated residual function and the seed vector $\delta \vec{W}^n$, which is a solution update in terms of the conservative variables $\vec{W}$. In a line-search terminology, the update step $\delta \vec{W}^n$ corresponds to the search direction vector usually denoted p^n [103].

Next step is to assess second Wolfe condition (line 11 in Algorithm 4). If the condition is satisfied, the value of CFL number is increased by factor $s^{up} = 1.1$, otherwise it is decreased according to the formula shown in line 14 of Algorithm 4. Additionally, the new CFL number is multiplied by the ratio of the old and current residual norm, which allows for a larger CFL when the convergence slope improves (reduction in residual norm) or reduces the CFL further when the solver starts diverging - see line 16 in Algorithm 4. This step essentially alters the variable s and plays a role of additional safeguarding for too fast or too slow change in the CFL number.

Since the accuracy of the intermediate solution $\vec{U}^n$ is not of primary concern, only a single iteration of line search is performed after each solver cycle. This prevents costly evaluations of the second Wolfe conditions shown in line 11 of Algorithm 4. As the CFL number becomes very large, no further improvement is usually noticed in the convergence slope or wall-clock time [122]. In order to prevent costly calculations of the line search algorithm, the adjustment routine is executed only when the current CFL is below the upper limit defined as $\mathrm{CFL}^{max} = 10^8$.

The influence of automatic CFL adjustment on the solver run-time is investigated using three test cases described in Section 4.3. The results are presented in Section 4.4.2.

4.2.3 Jacobian re-computation control

The STAMPS workhorse solver JT-KIRK [18] uses an ILU-preconditioner that is based on Jacobian of a 1^{st}-order discretisation scheme. As the flow solution evolves and approaches stationary point, the 1^{st}-order Jacobian (Eq. 4.2) and hence ILU-preconditioner do not experience much variation. In such a case, the re-computation of both can be switched off and computational effort of the solver reduced. However, quantifying the criterion that determines when the preconditioner from the previous solver cycle can be re-used is not straightforward. A computationally cheap method based on analysis of residual norm fluctuations between solver cycles (convergence curve smoothness), and slope analysis of the residual norm is proposed in this work. It is based on simple observation of typical behaviour of residual norm changes as a solver converges to a stationary point (see Figure 4.1).

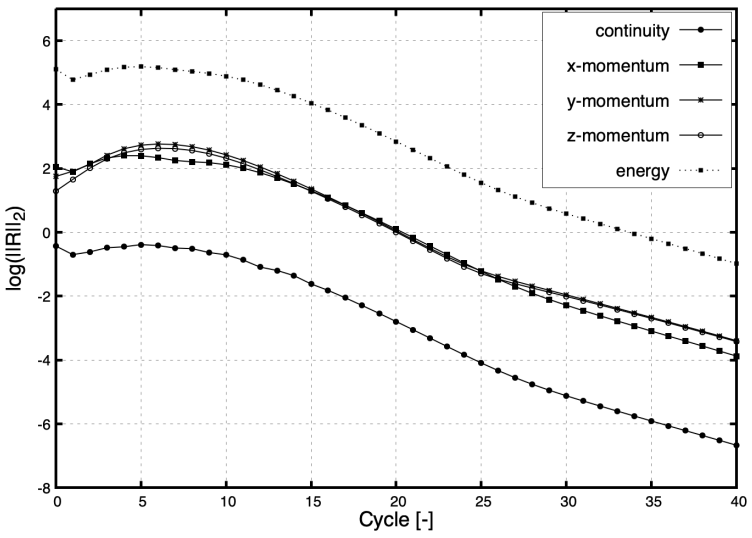


Figure 4.1: Example of residual norm convergence history

In the first 5 solver cycles the residual norm increases, and some fluctuations in norm value are visible until around 10^{th} iteration. However, after around 10 to 15 solver cycles the rate of change of residual norm starts to settle, and after 25 iterations the convergence curves has nearly constant slope. Furthermore, the convergence curves becomes smooth - near linear on a logarithmic scale plot with base 10. If the norm of residual reduces, and fluctuations in the norm value between solver cycles are low the algorithm switches off the re-computation of

Jacobian and preconditioner while controlling the mentioned conditions at each solver iteration. Although this set of conditions is not rigorous enough to provide high confidence for switching off the re-computation of the Jacobian, it is very cheap computationally and may lead to reduction in run-time of a solver. In this work, the proposed conditions for controlling Jacobian re-computation will be assessed using a set of practical CFD applications.

Note that for simplicity, in the following part of this work the *Jacobian/preconditioner re-computation control* will be often referred to as the *Jacobian re-computation control*.

Algorithm 5 summarises all the conditions that are assessed to determine whether the Jacobian/preconditioner has to be updated or not. Additionally, if the CFL-adjustment algorithm is active (see Algorithm 4) the re-computation of Jacobian is switched off only when the CFL number is increased between the previous and next iteration. Alternative state of the art approaches used to reduce (lag) number of re-computation of Jacobian and preconditioner were presented by Brown and Brune [77] or Knoll and Keyes [78].

Algorithm 5 System Matrix Re-computation Control (**A-Control**)

$Y_{i_{eq}}^{i_{probe}} \leftarrow$ `store_xy`$(N_{PWS}, log(||\vec{R}^n||_2))$
$\mathbf{A}_{recompute} \leftarrow$ True
if $(||\vec{R}^n||_2 < ||\vec{R}||_2^{min})$ **and** $(i_{Cycle} \geq N_{PWS})$ **and** $(CFL > 10^4)$ **then**
    $\mathbf{A}_{recompute} \leftarrow$ False
    for $i_{eq} = 1, N_{eq}$ **do**
        $(a_{i_{eq}}, b_{i_{eq}}) \leftarrow$ `LSQlinefit`$(N_{PWS},\ Y_{i_{eq}}^{N_{PWS}})$
        $\delta\widetilde{Y}_{i_{eqn}} \leftarrow$ `meandeviation`$(a_{i_{eq}},\ b_{i_{eq}},\ N_{PWS},\ Y_{i_{eq}}^{N_{PWS}})$
        if $(a_{i_{eq}} \geq 0$ **and** $\delta\widetilde{Y}_{i_{eqn}} > \delta\widetilde{Y}_{Limit})$ **then**
            $\mathbf{A}_{recompute} \leftarrow$ True
            `return` $\mathbf{A}_{recompute}$
        end if
    end for
end if

The variables and constants are first explained.

- $Y_{i_{eq}}^{i_{probe}}$ - exponent of the residual norm. It is stored for solver cycles and for each equation (i_{eq}).
- N_{PWS} - probing window size (PWS), which is number of solver cycles used for residual norm convergence analysis (user-defined input).
- $\mathbf{A}_{recompute}$ - a Boolean variable that controls re-computation of Jacobian and preconditioner.
- $||\vec{R}||_2^{min}$ - lowest residual norm within all solver cycles until the current iteration.
- N_{eq} - number of equations, for inviscid solver it is equal to 5.
- $(a_{i_{eq}}, b_{i_{eq}})$ - coefficients of a linear function from least square fitting.
- $\widetilde{Y}_{i_{eqn}}$ - average distance between the accumulated data set $Y_{N_{eq}}^{N_{PWS}}$ and linear function defined by coefficients $(a_{i_{eq}}, b_{i_{eq}})$. The mean deviation is calculated for each equation independently.

The procedure presented in Algorithm 5 is as follows. First, the residual norm exponents are accumulated using `store_xy` function and are stored in a variable $Y_{i_{eq}}^{N_{PWS}}$. The accumulation is done for each equation independently. A total of N_{PWS} entries is stored in the array. Next, the Boolean value of $\mathbf{A}_{recompute}$ is initialised with True (re-compute Jacobian), and a set of preliminary conditions is assessed:

- The first condition in line 3 checks if the number of solver cycles is greater or equal to the size of probing window.
- The second condition, checks if the current residual norms fall below the lowest residual norm recorder until the current solver cycle.
- The final condition is to analyse the current value of CFL number. The condition is met when the CFL value is greater than 10^4 that serves as an additional safety margin.

If all the conditions are satisfied, the Boolean variable is set to False and the algorithm executes analysis of convergence curve slope and smoothness (line 5-12). The analysis is performed for each equation independently and only when all conditions are satisfactory the re-computation of the Jacobian remains switched off ($\mathbf{A}_{recompute}$ =False).

First the coefficients of a linear function are obtained using least squares fitting function (`LSQlinefit`) as shown in Figure 4.2. Next, the average distance between the data set $Y_{N_{eq}}^{N_{PWS}}$ and linear function defined by coefficients $(a_{i_{eq}}, b_{i_{eq}})$ is calculated. Finally, the two conditions are assessed: a) slope of least squares fitted line (must be negative to keep $\mathbf{A}_{recompute}$ =False), and b) the 'smoothness' of the convergence curve (a user defined limit is set $\delta\widetilde{Y}_{Limit} = 0.05$). Only when the residual norm is decreasing and convergence curves are smooth for each equation the Jacobian re-computation is switched off.

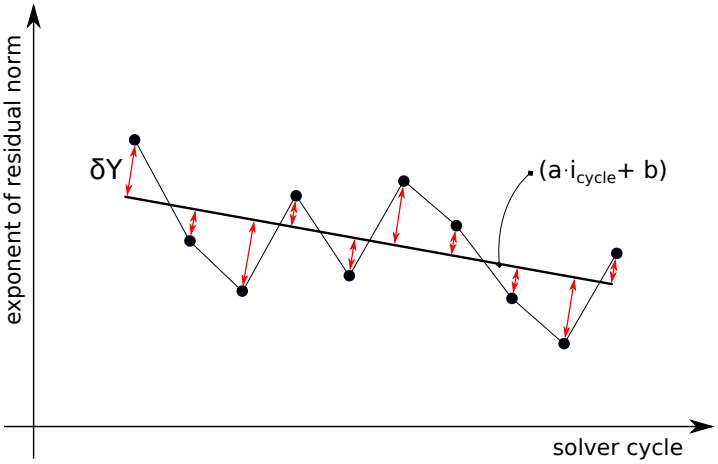


Figure 4.2: Linear least squares function fit and definition of δY (`LSQlinefit` function)

In the current work, the choice of 'smoothness' criterion that controls Jacobian re-computation originates from the fact that a 'noisy' convergence history of residual norm usually indicates high non-linearities in the system of flow equations as the solution evolves. When the oscillations are strong it is not safe to switch off preconditioner re-computation for the next solver cycle.

The STAMPS solver run-time savings due to the reduced re-computation frequency of the ILU-preconditioner and Jacobian are presented in Section 4.4.2.

4.3 Test cases and setup

The three cases presented in Figures 4.3—4.5 are used in order to test the implemented stabilisation methods and other code enhancements described in sections 4.2—4.6. The default solver settings (as described in Section 2.1) are used for each case, i.e., 2^{nd}-order discretisation accuracy, ROE flux scheme, JT-KIRK solver, VEM limiter, and Cell-based gradient. Key information about each test case is provided.

2D RAE2822 aerofoil

Key information for RAE2822 case (Figure 4.3):

- Mesh type/size: hexahedral / 23 458 nodes.
- Max control volume aspect ratio: 597.
- Flow: subsonic, turbulent (SA).
- Flow conditions: Ma $= 0.2$, Angle of attack $AoA = 0^{\circ}$, $T_{\infty} = 300K$, $p_{\infty} = 101325Pa$.
- Initialisation: Freestream conditions.
- Multigrid: 3 mesh levels.
- Linear solver: $GMRES(10)$, where number 10 in parentheses stands for the number of Krylov vectors.

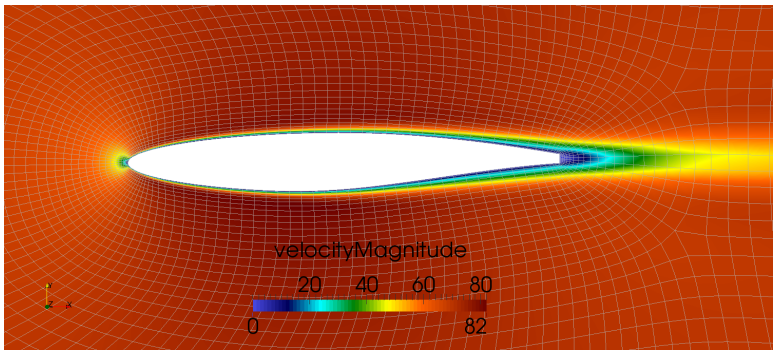


Figure 4.3: Subsonic RAE2822 - mesh and flow

3D Onera M6 wing

Key information for Onera M6 wing case (Figure 4.4):

- Mesh type/size: tetrahedral / 135 204 nodes.
- Max control volume aspect ratio: 4.8.
- Flow: transonic, inviscid.
- Flow conditions: Ma $= 0.8395$, $AoA = 3.06$, $T_\infty = 300K$, $p_\infty = 101325Pa$.
- Initialisation: Freestream conditions.
- Multigrid: 2 mesh levels.
- Linear solver: $GMRES(10)$.

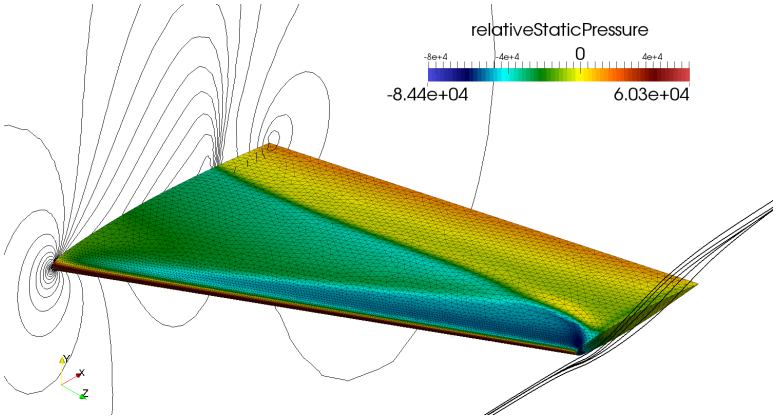


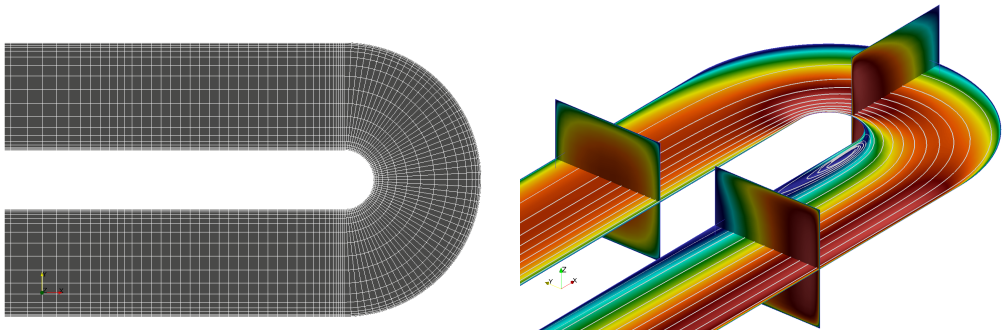
Figure 4.4: Transonic M6 - mesh and flow

3D U-bend

Key information for a low Mach number U-bend case (Figure 4.5):

- Mesh type/size: hexahedral / 191 700 nodes.
- Max control volume aspect ratio: 3065.
- Flow: subsonic (low Mach number), turbulent (SA).
- Flow conditions: Ma $= 0.02566$, $T_\infty = 293.15K$, $p_\infty = 101300Pa$.
- Initialisation: zero velocity (to prevent reverse flow in the half of the domain).

- Multigrid: 3 mesh levels.
- Linear solver: $GMRES(30)$.



(a) Mesh section (u-turn zoom) (b) Flow

Figure 4.5: Subsonic U-bend channel

Measurement consistency of a solver run-time

For all the test cases, STAMPS was run on a single core (*number* 0) always using the same machine to achieve a consistent measurement of the run-time of the solver. The usage of the CPU core (0) was achieved with the Linux tool **taskset**:

```
taskset -c 0 STAMPS settings.json 1
```

Summary info about the used CPU are obtained with the terminal command `cat/proc/cpuinfo`:

```
model name : Intel(R) Xeon(R) CPU E3-1240 v3 @ 3.40GHz
cpu family : 6
model : 60
cpu MHz : 3401.000
cache size : 8192 KB
cpu cores : 4
```

4.4 CFL-adjustment and Jacobian re-computation control

This section describes results obtained for the enhancements of STAMPS's solver. The enhancements have been already introduced in Section 4.2. First the nomenclature is provided, then the results are presented.

4.4.1 Nomenclature

The nomenclature that will be used in graphs with results is explained below:

- **BASELINE** - A standard JT-KIRK implicit solver with fixed CFL number. The CFL number was set to the maximum value that allows for the stable run (with ∼5% precision), and it differs for each case:
  - RAE aerofoil: $CFL_{BASE} = 350$
  - M6 aerofoil: $CFL_{BASE} = 20$
  - U-bend aerofoil: $CFL_{BASE} = 1000$
- **Restarted 100 × $\mathbf{CFL_{BASE}}$** - This label is be used to denote a typical engineering strategy for solving flow equations without a built in automatic CFL number adjustment. First, the CFL_{BASE} is used to run a few solver iterations and achieve a better initial guess of the solution. Next, the solver is interrupted and the CFL number manually adjusted (increased 100 times). The solver is then restarted using the solution from the pre-initialised run with CFL_{BASE}. The purpose of the 'restarted' run is to compare the manually altered CFL number with the automatic CFL adjustment approach implemented in STAMPS. The number of initialisation iterations was set to 10 for RAE aerofoil and M6 wing cases. 50 iterations were required for a stable restart of the U-bend case.
- **CFL-RMPG** - CFL-adjustment active, where the initial CFL number is set as for the **BASELINE** run.
- **Restarted 100 × $\mathbf{CFL_{BASE}}$ | A-CTRL** - System matrix re-computation control active. The same restart scenario was used as for the **Restarted**

100 × CFL$_{\mathbf{BASE}}$ case to see directly how much time can be saved when the Jacobian and ILU-preconditioner are not re-computed at each solver cycle.

- **CFL-RMPG | A-CTRL** - CFL-adjustment and Jacobian re-computation control active.

4.4.2 Results

Results presented in this section show the influence of CFL-adjustment and Jacobian re-computation control on the wall-clock time of the STAMPS solver. The three test cases described in Section 4.3 were run and the residual norm convergence histories are shown in Figures 4.6, 4.8, 4.10.

RAE2822

The convergence of the density residual norm (non-normalised) for the 2D RAE aerofoil case is presented in Figure 4.7. The shortest overall wall-clock times are obtained for the **Restarted 100 × CFL$_{\mathbf{BASE}}$** | case and when the Jacobian re-computation control is activated. The **A-CTRL** allowed for around 16% reduction in the run-time which can be seen when comparing convergence curves with the hollow-square and filled-square markers.

A similar solver run-time is recorded when the CFL-adjustment and Jacobian re-computation control are active. The difference between this case and the case with only Jacobian re-computation control is due to the additional computational cost associated with the line search algorithm (see Algorithm 4) of **CFL-RMPG**.

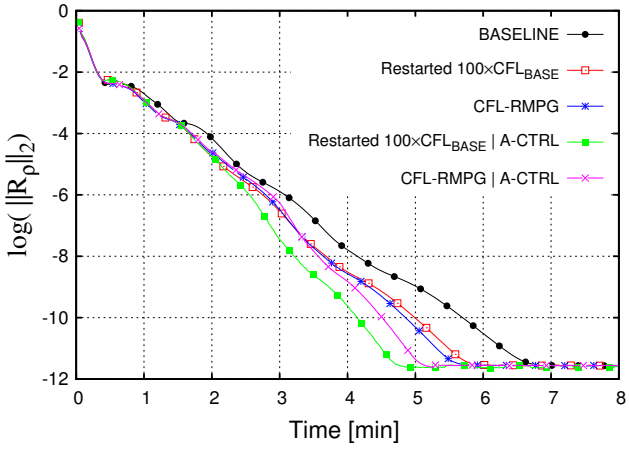


Figure 4.6: CFL-adjustment and Jacobian re-computation control - RAE aerofoil

Figure 4.7 shows the CFL variable (bottom curve, left axis) and $\mathbf{A}_{recompute}$ switch (upper curve, right axis) over time for the case with **A-CTRL** and **CFL-RMPG** active. In the case of Jacobian re-computation control, the value (1) indicates that the Jacobian and ILU-preconditioner will be recalculated for the next solver cycle, and the value (0) is switched off. For the RAE case, Jacobian and preconditioner are being re-computed for the first 3 minutes of the solver run-time. The remaining 2 minutes (before the solution is converged) the Jacobian re-computation is switched off. Hence the frequency of Jacobian and preconditioner re-computation was reduced by around 40%.

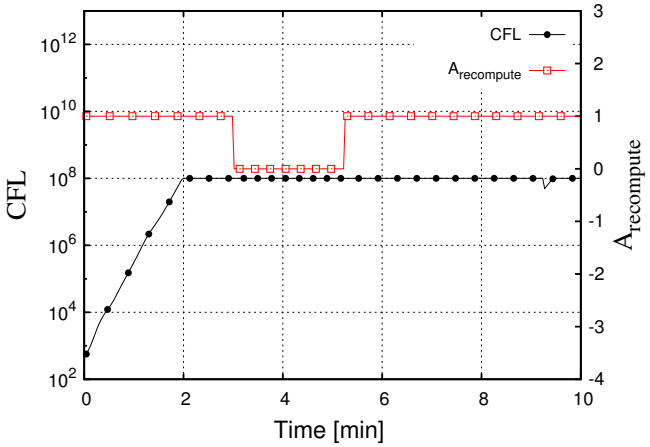


Figure 4.7: History of CFL number change and activity of Jacobian re-computation wrt. time for (**A-CTRL** | **CFL-RMPG**) RAE aerofoil case

As explained in Section 4.2.2 the CFL number is no longer increased above the value $CFL^{max} = 10^8$. Note that after around 5 minutes when the residual norm is not experiencing more reduction the Jacobian/preconditioner re-computation is switched on again. This is because of the condition from line 3 in Algorithm 5 which is in place for the safety margin of empirically derived criteria. However, as the solution is already converged the solver would be terminated, and activation of Jacobian/preconditioner re-computation at this stage is not relevant for the solver run-time. The termination criteria are currently not implemented in the STAMPS solver.

Onera M6 wing

Similar conclusions can be made for the M6 wing case - see Figure 4.8. However, for this case, the shortest wall-clock time is obtained for both methods **A-CTRL, CFL-RMPG** active. This is due to the very low initial CFL number used which was necessary for the stable run of the **BASELINE** case. The instabilities were most likely associated with the formation of shock waves (Figure 4.4).

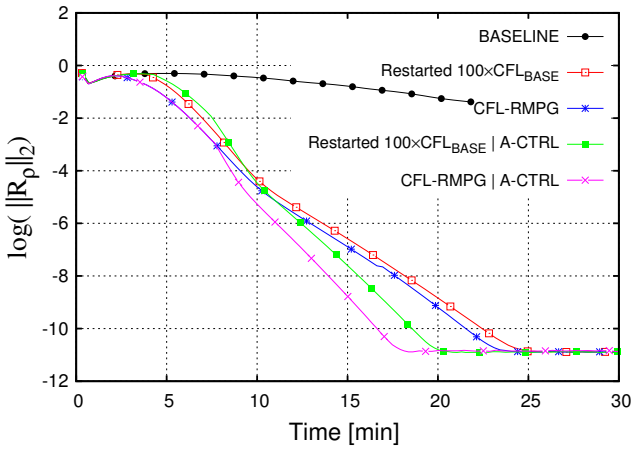


Figure 4.8: CFL-adjustment and Jacobian re-computation control - M6 wing

Figure 4.9 shows that the CFL number is smoothly increasing with the evolving solution and stopped at the level around 10^4. The termination of ramping before reaching the CFL^{max} limit is due to the **A-CTRL** parameter value being (0) (no Jacobian re-computation). In the current implementation, the CFL number is becoming fixed; otherwise the diagonal term of the the system matrix **A** Eq. 4.2 would need to be replaced and the ILU-preconditioner re-computed which would add to the computational cost. The limit of **A-CTRL** algorithm activation is set to half the order of the CFL^{max}, i.e. 10^4, to prevent premature termination of the ramping - see Algorithm 5 in Section 4.2.3.

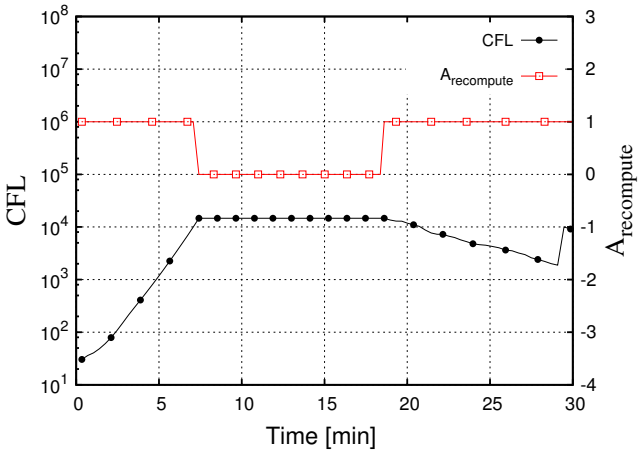


Figure 4.9: History of CFL number change and activity of Jacobian re-computation wrt. time for **A-CTRL | CFL-RMPG** M6 wing case

U-bend

Figure 4.10 shows results for the U-bend channel case. All the cases where either of the techniques was used show a faster convergence. However, the run-time reduction is lower as compared to the RAE or M6 wing cases, which is because more GMRES vectors (30) were used for this case - see case description in Section 4.3. Hence, relatively more time is spent in the GMRES linear solver than on the system matrix and preconditioner computation (per single solver cycle). A larger number of Krylov vectors is required for the U-bend case (especially in the initial convergence stage), as it is a challenging case with high aspect ratio cells and a turbulent flow with separation, resulting in very slow convergence.

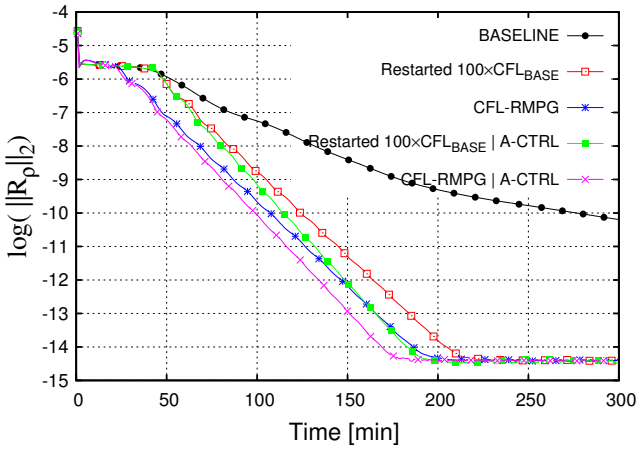


Figure 4.10: CFL-adjustment and Jacobian re-computation control - U-bend channel

The automatic adjustment method allows the CFL number to be quickly increased towards CFL_{max} - Figure 4.11. However, the system matrix re-computation is triggered on/off several times. This is due to the 'wavy' convergence curve character (zoom Figure 4.10) which results in the condition from line 8 in Algorithm 5 to trigger the $\mathbf{A-recompute}$ on and off several times. The 'wavy' convergence pattern is caused by the low Mach number flow condition, which is known to be challenging for the compressible solvers [137]. As will be shown in Section 4.6.1, once the low Mach scaling (LMS) was switched on the oscillations disappeared.

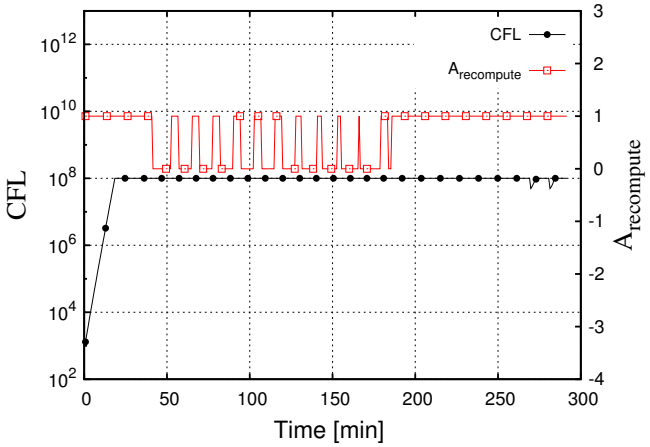
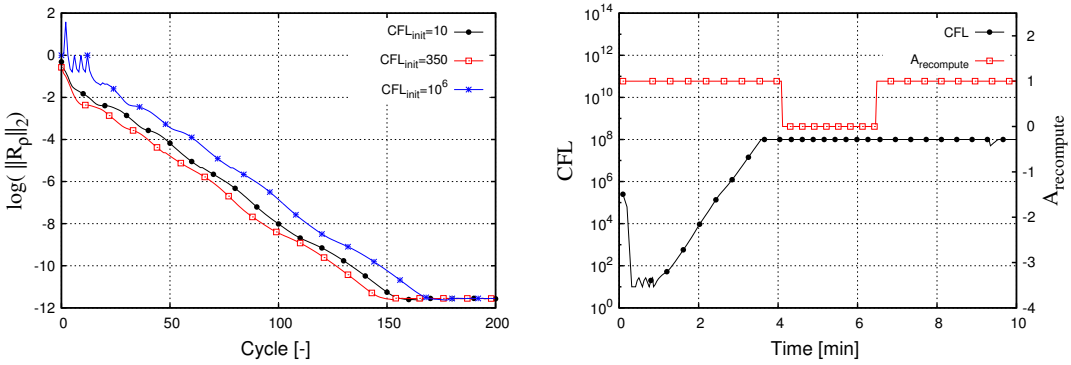


Figure 4.11: History of CFL number change and activity of Jacobian re-computation vs. time for (**A-CTRL** | **CFL-RMPG**) U-bend case

Sensitivity to initial value of CFL number

Although the adjustment of the CFL number is controlled by Algorithm 3 the initial value (CFL_{init}) is a user defined quantity. A study was performed to investigate the influence of the CFL_{init} on the number of cycles required to converge the solution. Ideally, the algorithm should show low sensitivity to the initial value of the CFL number. Results for the three test cases shown in Figs. 4.12-4.14 confirm that the initial CFL number has little effect on the number of cycles required to converge. However, for the RAE case around 15% more cycles are required for $\text{CFL}_{init} = 10^6$ to achieve the same convergence level as compared to the $\text{CFL}_{init} = 10$ or $\text{CFL}_{init} = 350$ - Fig. 4.12. This is because the large initial CFL number resulted in solution divergence within the first few cycles as indicated by the strong (single) spike in the convergence curve, Fig. 4.12a. The stable solution state was recovered after several iterations thanks to the $Not - a - Number$

safeguarding introduced in Algorithm 3. The CFL number was rapidly decreased to a value around 20 as shown in Fig. 4.12b. After the solution recovery several smaller spikes are visible in convergence curve 4.12a as well as in CFL number history 4.12b. This is a result of the solver trying to recover from the poor state obtained by using a too high initial CFL number.



(a) Residual norm convergence history

(b) CFL history for $CFL_{init} = 10^6$

Figure 4.12: Initial CFL study - RAE aerofoil

For the M6 wing (Figure 4.13) case and the U-bend case (Figure 4.14) a small reduction in the number of cycles is obtained as the initial CFL number value is increased. This is an expected behaviour of the algorithm for the cases where the high initial CFL number is not causing solution divergence.

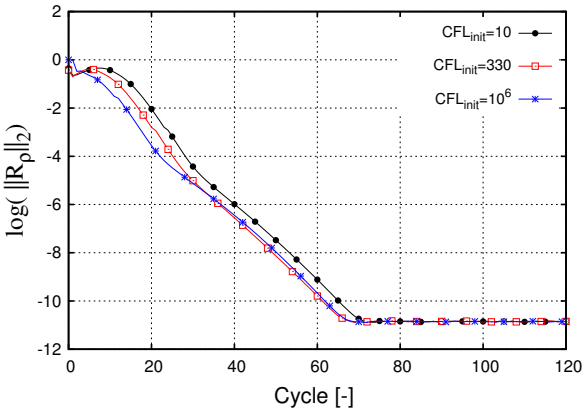


Figure 4.13: Initial CFL study - M6 wing

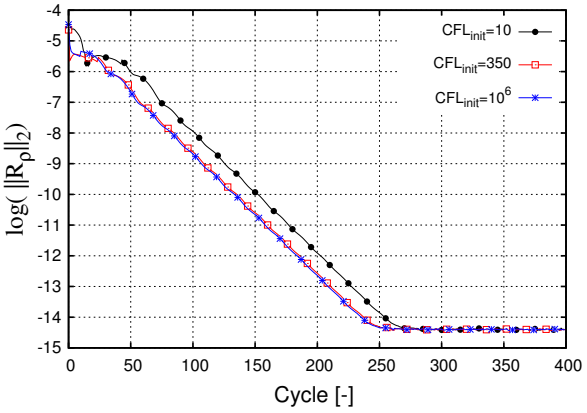


Figure 4.14: Initial CFL study - U-bend channel

4.5 Multigrid start-up

Multigrid start-up (MGS) was briefly introduced in Section 2.6. A sequence of coarse meshes used in geometric multigrid can be also exploited to provide a better initial guess for the solver. MGS is a simplified version of full multigrid start-up (FMG) and it is executed before starting the main solver. The idea is to first achieve a certain level of convergence on the coarsest grid, then prolong the obtained solution to finer grid level and repeat until the finest grid level is reached. First-order discretisation is employed for MGS start-up, and only a single grid is used for each level as opposed to FMG method which uses V-cycle on each grid level above the coarsest mesh.

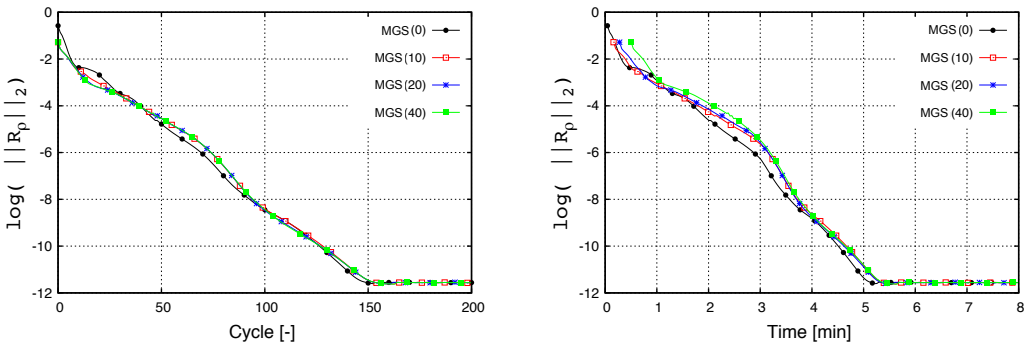
The number of iterations to perform on each coarse grid is defined by the user input. After the MGS initialisation is done the standard V-cycle is employed to obtain the solution on the primary/finest mesh.

4.5.1 Results

The effect of varying the number of MGS iterations was performed and results for MGS 0, 10, 20, and 40 iterations are presented in Figures 4.15—4.17. The value in parentheses in figures' legends indicates the number of MGS-cycles used at each coarse grid, and MGS(0) means that the MGS-initialisation was switched off. Only the finest grid residuals are shown in the figures. The CFL-adjustment and system matrix re-computation control were active for the test runs.

The shifted (to the right) initial convergence point visible in Figures 4.15—4.17 for the active MGS start-up, indicates the amount of time required for the coarse grid run (MGS-initialisation). As can be seen, the added computational cost of the full multigrid technique is recovered by the fact that a better initial state is obtained.

As the results show, the run-times of the STAMPS solver are almost identical for MGS with 0, 10, 20, and 40 iterations hence no benefits could be seen for using the methodology. However, based on the author's experience, an MGS 'hot' start allows for a stable run with a higher initial CFL number and hence should be used by default. Additionally, it is important to remember that MGS is a simplified version of FMG algorithm, hence the standard version that performs V-cycle could show more benefits and should be tested in future work.



(a) Residual norm convergence history vs. number of cycles

(b) Residual norm convergence history vs. run-time

Figure 4.15: MGS study - RAE aerofoil

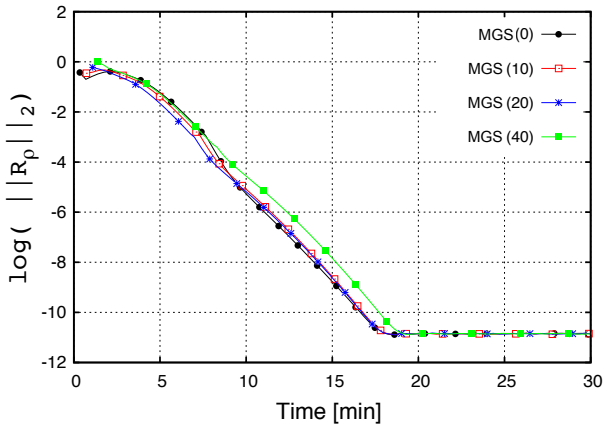


Figure 4.16: MGS study - M6 wing

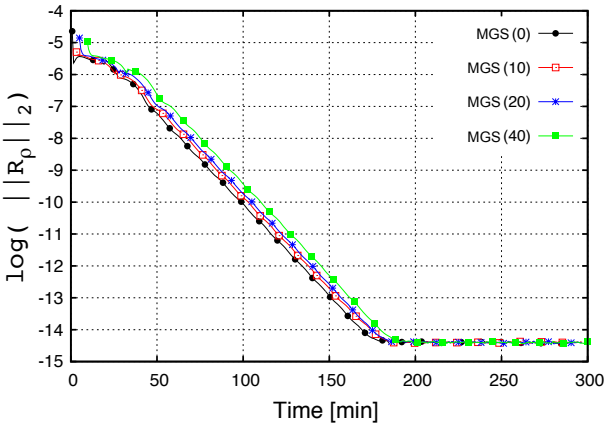


Figure 4.17: MGS study - U-bend channel

4.6 Convergence acceleration techniques

Three convergence acceleration techniques are presented in this section:

1. Low Mach scaling (LMS), Section 4.6.1, allows convergence for low Mach number flows to be enhanced with a compressible solver.

2. Residual-based time-stepping (RBTS), Section 4.6.2, scales the time term of the implicit system matrix (Eq. 4.13) based on the local residual norm. This allows the CFL number to adjust to the size of control volumes where larger time steps can be taken. Hence, it can be seen as a form of implicit local time-stepping.

3. Enhancement for highly stretched meshes (AR), Section 4.6.3, is useful when running turbulent simulations with high aspect ratio cells near the wall. It scales the local time step according to a control volume aspect ratio (AR).

4.6.1 Low Mach scaling (LMS)

For subsonic flow simulations where the Mach number becomes very low, the convective terms of the governing system of equations (see Eq. A.2) become stiff as a result of large differences between acoustic and convective eigenvalues: (V, $V + c$, $V - c$). This, in turn, can lead to convergence problems for compressible

solvers, as well as an incorrect scaling of artificial dissipation of the *ROE* flux scheme as the Mach number approaches zero [138].

A simple technique involves adjusting the ambient conditions (p, T) to increase the Mach number while keeping the Reynolds number unchanged and hence maintaining the same flow physics. This can be used as a form of low Mach preconditioning and was explored by Forsythe [137]. The method is implemented in STAMPS to improve convergence for low Mach number applications.

For demonstration purposes, the U-bend channel case described in Section 4.3 is used. Given the initial set of ambient conditions presented in Table 4.1 the new increased Mach number is imposed, which should be neither too high such that the compressible effects remain negligible, nor too low to prevent described issues with compressible solvers. The value of $\mathrm{Ma} = (0.1 - 0.2)$ meets both criteria. For the purpose of the U-bend example, $\mathrm{Ma}^{lms} = 0.2$ is used.

In order to maintain the same Reynolds number with an adjusted Mach number the acoustic/sound speed has to be modified as presented in Eq. 4.6, which is derived using Eq. 4.5. The variable L is a characteristic dimension, that is, a hydraulic diameter of the channel ($L = D_h = 0.075$ m). Other constants are provided in Appendix A.1.

$$\mathrm{Re} = \frac{\rho(V)L}{\mu} = \frac{\rho\ (\mathrm{Ma}\ c)\ L}{\mu}, \quad \mathrm{Ma} = \frac{V}{c} \tag{4.5}$$

$$c^{lms} = \frac{\mu \mathrm{Re}}{\rho L\ \mathrm{Ma}^{lms}} \tag{4.6}$$

Modified acoustic speed requires adjustment of ambient temperature and pressure according to Eqs. 4.7 and 4.8. Finally, a new set of ambient conditions is obtained without altering the Reynolds number as shown in Table 4.2.

$$c = \sqrt{\gamma RT} \tag{4.7}$$

$$p = \rho RT \tag{4.8}$$

In STAMPS, Southerland's law is used for the calculation of dynamic viscos-

ity - see Eq. A.22, Appendix A.2. However, the low-Mach-scaling methodology requires the viscosity to be constant to maintain the Reynolds number. As the Mach number is low (Ma = 0.2), the assumption of constant viscosity is valid and has a negligible effect on the flow solution. The viscosity is calculated using the original ambient state ($\mu^{T=293.15\ \mathrm{K}} = 1.605 \times 10^{-5} \frac{\mathrm{kg}}{\mathrm{m\ s}}$).

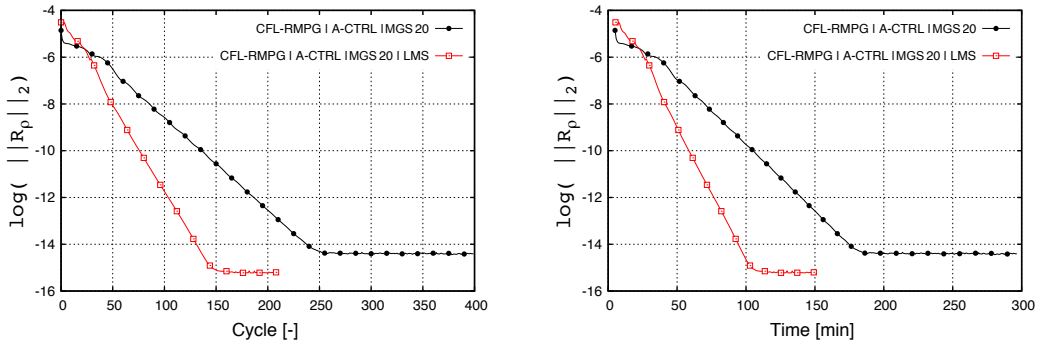
Variable	Value
Ma [−]	0.02566
c [m/s]	343.287
T [K]	293.15
p [Pa]	101 300
Re [−]	49 533.203

Table 4.1: Initial boundary conditions for low Mach number channel flow

Variable	Value
Ma^{lms} [−]	0.2
c^{lms} [m/s]	44.044
T^{lms} [K]	4.8255
p^{lms} [Pa]	1667.488
Re^{lms} [−]	49 533.203

Table 4.2: New ambient conditions (after scaling) for the U-bend channel

The U-bend case (see Section 4.3) is used to analyse how low Mach scaling (LMS) influences convergence of the compressible solver. The CFL-adjustment, A-CTRL, and a 20 MGS start-up iterations were used as the baseline for the comparison. The results show a significant reduction in run-time and the required number of cycles (nearly 50%) when the low Mach scaling is active - Figure 4.18.



(a) Wrt. time (b) Wrt. cycles

Figure 4.18: Low-Mach-scaling for U-bend case

Furthermore, the low-frequency noise present in the residual norm history of the baseline case (Figure 4.18) that is captured by the **A-CTRL** algorithm (see Section 4.2.2) is removed when low-Mach-scaling is active. As a result, once the A-CTRL parameter is triggered off it remains unchanged until the convergence is reached for the case with LMS active - Figure 4.19. This confirms that the 'wavy' character of convergence curve for the baseline case (discussed in Section 4.4.2) is related to the low Mach number flow regime.

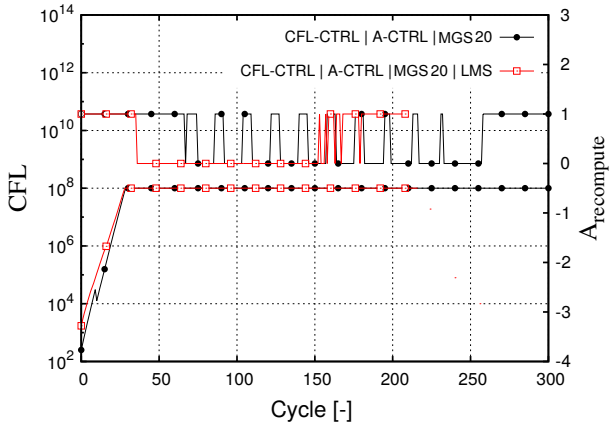


Figure 4.19: CFL and A-CTRL histories with and without LMS

The low-Mach-scaling technique acts as a low-Mach-preconditioner for the compressible system of equations. As a result of the decreased ratio of fastest to slowest eigenvalue of the system of governing equations for the set of conditions defined in Table 4.2, the solver run-time can be significantly reduced (around 50%) for the U-bend channel flow example.

4.6.2 Residual-based time-stepping (RBTS)

An alternative way of computing a local time step was proposed by Michalak [20] and applied to the inviscid flow solver (Eq. 4.9). The methodology uses the ratio of global residual norm to local residual norm in order to guide the selection of the new local time step as presented in Eqs. 4.9 and 4.10 (see Eq. 2.86 for the original definition of local time step Δt_i^n).

$$\triangle t_{i,RBTS}^n = \triangle t_i^n \, RBTS_i^n \tag{4.9}$$

$$RBTS_i^n = \texttt{max}\left(\texttt{min}\left(\frac{||R^n||_2}{||R_i^n||_2}, 10^2\right), 10^{-2}\right) \tag{4.10}$$

The L_2-norms in Eq. 4.10 are defined as shown in Eqs. 4.11 (global residual norm) and 4.12 (local residual norm at node i). The constant N is a total number of control volumes, and N_{eq} is a number of RANS equations ($N_{eq} = 6$ for the compressible N-S equations with SA turbulence model).

$$||R^n||_2 = \sqrt{\sum_{i}^{N}\left(\sum_{i_{eq}}^{N_{eq}}\left(R_{i,i_{eq}}^n\right)^2\right)} \tag{4.11}$$

$$||R_i^n||_2 = \sqrt{\left(\sum_{i_{eq}}^{N_{eq}}\left(R_{i,i_{eq}}^n\right)^2\right)} \tag{4.12}$$

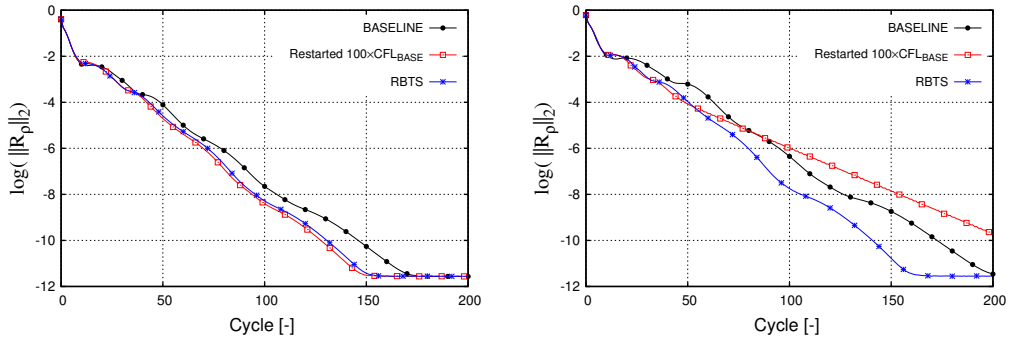
The system matrix $\mathbf{A}$ is hence modified as shown in Eq. 4.13.

$$A = \left[\frac{\Omega_i}{CFL^n \, RBTS_i^n \, \Delta t_i^n} + \left(\frac{\partial \vec{R}}{\partial \vec{W}}\right)_i^n\right] \tag{4.13}$$

The local residual tells how much imbalance is left in the equations due to the non-converged state. The imbalance is expected to be high in the areas of the computational domain where the current state is far from the solution, and low otherwise. As the solution is close to the stationary point (linear convergence regime) the large time step can be safely selected and the Newton step recovered. The time step definition proposed in Eq. 4.9 allows the above to be achieved, that

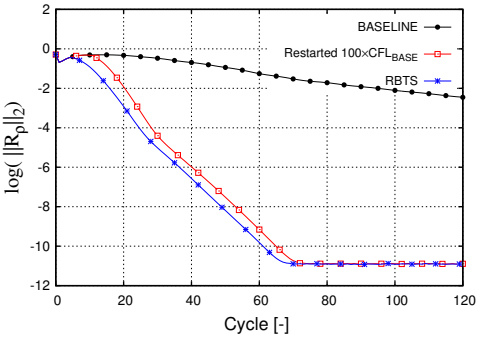
is, when the local residual is low the variable $RBTS_i^n$ gets large and $\mathbf{A}_i \leftarrow \left(\frac{\partial R}{\partial W}\right)_i$, otherwise the low $RBTS_i^n$ increases the time-step-related term in Eq. 4.13, and allows the solution to be evolved in a finite size pseudo time step (to stabilise the Newton-type solver). Furthermore, the variable $RBTS_i^n$ is limited to prevent convergence stall when the local residual gets very high, or to prevent instability of the implicit solver due to an excessive increase of time step (Eq. 4.9) when the local residual gets very low. The bounds are set between $[10^{-2}; 10^2]$ [20].

Results in Figures 4.20a—4.20d show the influence of the residual-based local time-stepping on the number of cycles required to converge as compared to the baseline and restarted case. In most cases the active RBTS parameter results in the same solver performance as for the case with an increased CFL number - see **RBTS** case vs. **Restarted 100** $\times$ $\mathbf{CFL_{BASE}}$ case for the RAE(Ma=0.2) (4.20a), M6 wing (4.20c), and U-bend (4.20d). However, for the RAE case run at the higher Mach number (Ma=0.3) it was found that an increase of the CFL number does not lead to the reduction of a number of cycles till convergence, whereas the activation of RBTS enhancement allows the same convergence level to be reached faster as compared to the baseline and restarted run - Figure 4.20b. The origin of the achieved enhanced convergence is a better condition number of the system matrix resulting from the RBTS-scaling.

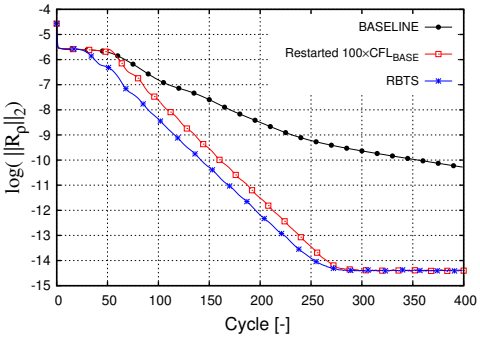


(a) RAE Ma=0.2

(b) RAE Ma=0.3



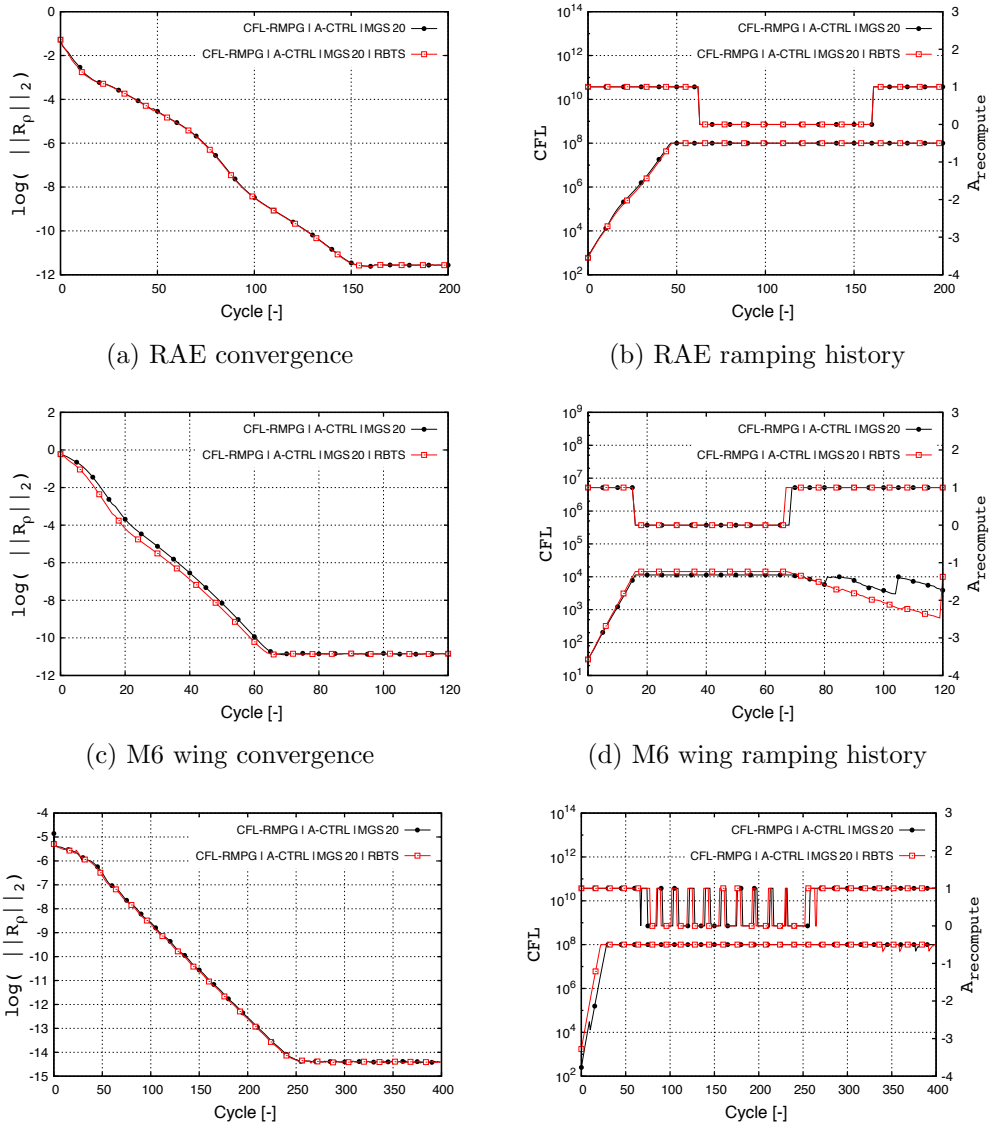
(c) M6 wing



(d) U-bend

Figure 4.20: Residual-based local time step - results

The residual-based local time-stepping was also run in combination with CFL-adjustment, A-CTRL, and MGS(20) start-up. The results in Figure 4.21 show almost no influence on the number of cycles required for the 'full' convergence. However, as the RBTS technique is shown to lead to a more rapid convergence for some cases (e.g. RAE(Ma=0.3) and not to affect the overall solver performance for others (RAE(Ma=0.2), M6 wing, U-bend), it is recommended as a default in STAMPS.



(a) RAE convergence

(b) RAE ramping history

(c) M6 wing convergence

(d) M6 wing ramping history

(e) U-bend convergence

(f) U-bend ramping history

Figure 4.21: Residual-based local time step with other solver enhancements

4.6.3 Enhancement for highly stretched meshes (AR)

Accurate simulation of turbulent flow often requires a very high mesh resolution near the wall to capture physics of the boundary layer. As the refined grid is required only in the wall normal direction, the high AR cells are obtained - see U-bend channel example in Figure 4.22. Aspect ratio of a mesh cell is defined as the ratio of longest edge to the shortest edge.

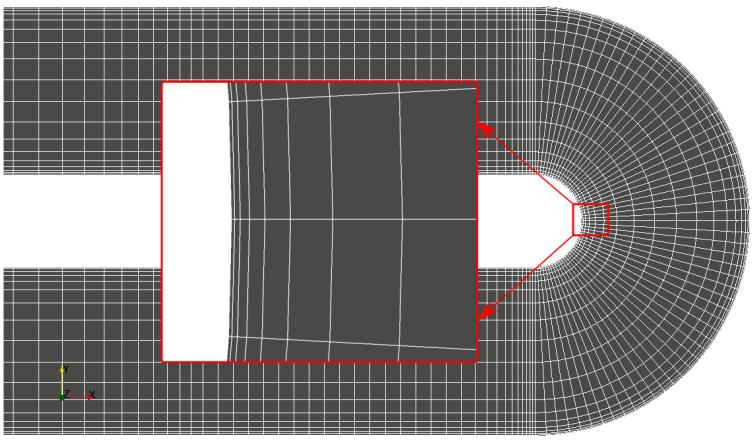


Figure 4.22: High aspect ratio (AR) cells near the wall - U-bend test case

The local pseudo time step shown in Eq. 4.14 is proportional to cell volume. High aspect ratio cells have very small volumes which leads to small local pseudo time steps. Consequently, the convergence of state vector $\vec{U}$ is expected to be very slow in high AR cells.

$$\Delta t_i^n = CFL \frac{\Omega_i}{\Lambda_c^n + \Lambda_v^n} \tag{4.14}$$

For the typical industrial meshes with a target $y+$ below 1, the aspect ratio can vary several orders of magnitude within the computational domain, where the values between $[1 : 1000]$ are common. This fact will cause the information to be transferred three orders of magnitude slower in the boundary layer due to the decrease in pseudo time step.

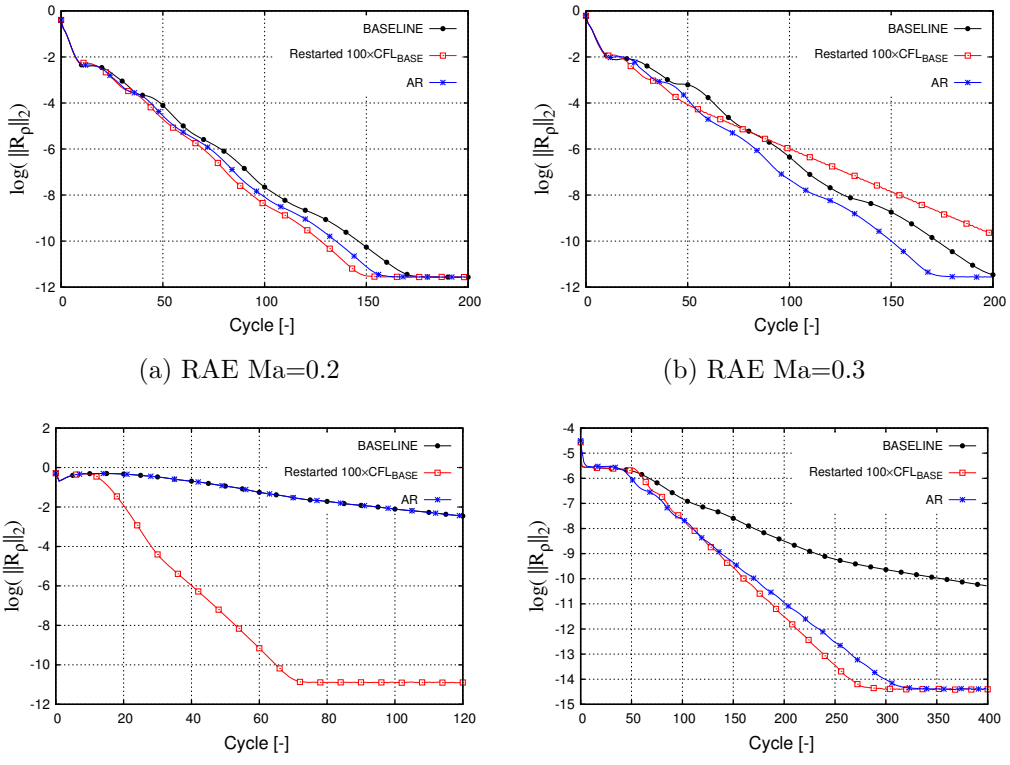
In order to prevent convergence slowdown, a more suitable pseudo time step definition can be used for highly stretched grids as shown in Eq. 4.15), where the variables $l_{i,min}$ and $l_{i,max}$ are the minimum and maximum edge length within the nearest neighbours of node i. This approach was proposed for example in ANSYS Fluent commercial solver. The new definition varies the time step from one cell to another based on the cell aspect ratio (AR) as compared to the standard definition from Eq. 2.86. If the cell is close to equilateral the aspect ratio is near unity and the time step remains unchanged. On the other hand, when the cells become highly-stretched the aspect ratio becomes large and the AR-scaling increases the original time step value.

$$\triangle t_{i,AR} = \triangle t_i \; AR_i, \quad AR_i = \frac{l_{i,max}}{l_{i,min}} \tag{4.15}$$

The system matrix **A** is hence modified as shown in Eq. 4.16.

$$\mathbf{A} = \left[\frac{\Omega_i}{CFL^n \, AR_i \, \Delta t_i^n} + \left(\frac{\partial \vec{R}}{\partial \vec{W}} \right)_i^n \right] \tag{4.16}$$

An influence of the aspect-ratio (AR) scaling on the number of solver cycles required to converge is very similar to the results obtained for the residual-based local time-stepping discussed in Section 4.6.2. For most tested cases the AR-scaling has only a minor influence on the convergence rate (Figures 4.23a, 4.23d), except for the RAE(Ma=0.3) example shown in Figure 4.23b. As the maximum cell AR for the M6 wing case is lower than 10, there is no noticeable change in convergence slope compared with the baseline case - Figure 4.23c. Note that all the figures in this section are presented only in terms of number of cycles because using run-time would lead to identical results - there is almost no additional cost for using the AR technique.



(a) RAE Ma=0.2

(b) RAE Ma=0.3

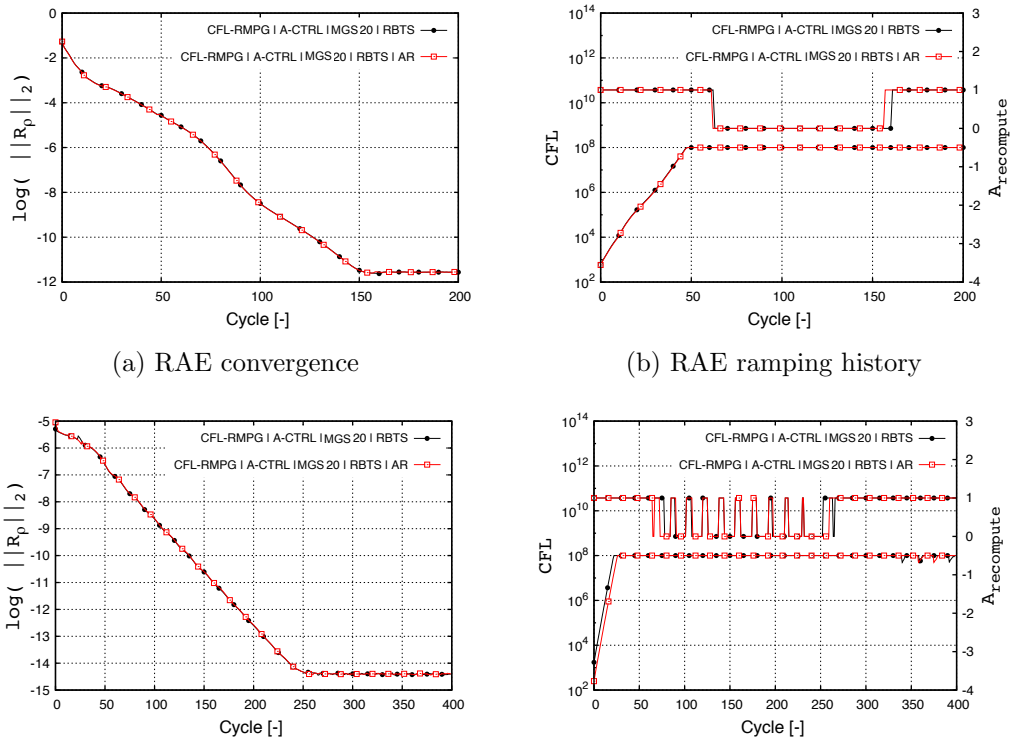
(c) M6 wing

(d) U-bend

Figure 4.23: Aspect ratio scaling of local time step - results

AR-scaling was also run in combination with CFL-adjustment, A-CTRL, MGS(20) start-up, and RBTS. The results in Figure 4.24 show the almost iden-

tical number of cycles required to converge. However, as AR-scaling has been proved to lead to a more rapid convergence for some cases and not to affect the overall solver performance for others, it is, like the RBTS technique, recommended as a default in STAMPS.



(a) RAE convergence

(b) RAE ramping history

(c) U-bend convergence

(d) U-bend ramping history

Figure 4.24: AR scaling with other solver enhancements

When RBTS and AR are both active, the first term in the system matrix $\mathbf{A}$ is calculated according to Eq. 4.17.

$$\mathbf{A} = \left[\frac{\Omega_i}{CFL^n \, RBTS_i^n \, AR_i \, \Delta t_i^n} + \left(\frac{\partial \vec{R}}{\partial \vec{W}} \right)_i^n \right] \tag{4.17}$$

Chapter 5

Mesh adaptation

This chapter is organised as follows. First the introduction and literature review on mesh adaptation techniques in CFD is presented in Section 5.1. Next, Section 5.2 presents the nomenclature, mathematical derivations of an output error, and the methodologies used for the truncation error and output error estimations. Section 5.3 describes the formulation of final adaptation sensors used to drive the refinement process. Testing of the proposed error estimation method is performed in Section 5.4, where the known manufactured solution (Appendix A.5) in a cube domain is used. The adaptation results using the re-meshing technique with BoxerMesh (Section 5.5) were obtained using the Rolls-Royce (RR) proprietary code Hydra during a 3-month secondment at RR. The refinement results with mmg3d, which are presented in Section 5.6, were obtained using the QMUL in-house solver STAMPS. The same adaptation sensors were developed and verified by the author in both codes.

5.1 Introduction

The adjoint method [17] is well-established as the most efficient technique for an aerodynamic shape optimisation with CFD. It allows shape sensitivities to be calculated at a cost that is several orders of magnitude lower as compared to, for example, finite-difference or tangent-linear approaches. Although adjoints make shape optimisation affordable, its typical cost is still at least an order of magnitude higher than a single flow evaluation. As discussed in chapter 4, using a

one-shot approach with optimally tuned parameters can reduce the optimisation cost to the factor of 5-10 times the cost of a primal solution. Further reductions can be obtained by improving robustness of the flow solver, and by minimising the user effort required for manual tuning of solver parameters which are application-dependent. Results presented in chapter 4 show that run-time of the flow solver can be reduced by 10-80% percent when the automatic CFL adjustment technique is used along with other enhancements introduced in the STAMPS solver. Larger reductions can be achieved for more complex shapes and flow conditions, which can be of particular importance for industrial cases. Using mesh adaptation is another effective way of reducing the computational cost of shape optimisation.

An important aspect of mesh adaptation is to develop an effective indicator of local errors in the quantity of interest i.e. the objective function. Adjoint sensitivities that are computed during the shape optimisation process can also be used to obtain a robust adaptation sensor. This sensor is used to drive a solution-adaptive mesh refinement that computes a more accurate objective function at the lower computational cost compared to error-estimators without adjoint weighting or compared to heuristic sensors. The adjoint-weighted truncation error was first proposed by Becker and Rannacher [29, 30] for incompressible Navier-Stokes equations (finite element discretisation) and applied to various 2D cases such as flow around cylinder or heat driven cavity. Similar analysis was presented by Giles and Pierce [25, 27] for finite volume discretisation, where authors shown how the adjoint-weighted truncation error can be used to improve prediction accuracy of functionals such as drag or lift.

More recently, due to the increased interest in adjoint method, the adjoint-weighted truncation error or output error gained popularity as an effective driver for an adaptation process [31, 32, 33, 34]. Through the adjoint weighting, the mesh adaptation is effectively targeted to those areas of the computational domain where the objective function is highly sensitive to mesh resolution, and the local error estimate is large. That is the key advantage of an output-based indicator as compared to other approaches such as, e.g., gradient/Hessian-based sensors [72, 49, 73] or pure truncation-error-based sensors [74, 141, 79]. While the latter at least attempt to estimate the actual errors, both of these methods apply the

refinement to all errors, regardless of whether they are relevant to the computation of the objective function or not. A cheaper yet effective alternative to the output sensor that is based on entropy variables was proposed by Fidkowski [80, 81]. In this work, however, the author focuses on the output-based adaptation indicator.

Assuming that the adjoint solver is available, the remaining challenge in obtaining an output-based sensor lies in estimating the truncation error. A popular approach in the literature has been presented by Venditti and Darmofal [82, 83]. The computational grid is locally refined preserving its original topology and the current solution interpolated to the refined grid is used to estimate the truncation error on the computational grid. Further refinement and application of this approach can be found in [31]. Even though no full solution is computed on the refined mesh, some work on higher-order interpolation has to be done on the refined grid, and the computational cost as well as memory usage is not negligible.

As an alternative, we could consider using the difference between the finest computational grid and a coarsened grid to estimate the local error as proposed by Fraysse et al. [84, 85] (τ-estimation). In this variant the computational effort is significantly reduced and, in the case of fully coarsened geometric multigrid methods, can even be at no additional effort since the coarse grid is already computed. On the other hand, the error estimate is then only valid for the coarse grid, and care has to be taken to make a valid extrapolation to the fine grid. Ponsin et al. [34] extended the τ-estimation method by weighting it with adjoint solution calculated on a coarse grid to obtain output-based sensor. However, in both τ-estimation approaches by Fraysse and Ponsin multigrid meshes are topologically consistent, and a coarse mesh is created by fusion of several fine mesh cells. This situation does not represent a general case for geometric multigrid solvers.

In this work, truncation error estimation is performed using building blocks of a geometric multigrid solver where in a general case the grid levels are topologically inconsistent. Multigrid meshes are generated using hip tool which performs unstructured grid coarsening based on an edge collapsing algorithm [37]. The proposed truncation error estimation method is more general than other approaches proposed in the literature, and can be easily implemented in geometric multi-

grid solvers such as an example Rolls-Royce proprietary code Hydra or QMUL in-house code STAMPS. The results shown in this chapter confirm that the approach can be successfully used for practical engineering applications.

An accurate error estimation method is an essential requirement for a successful mesh adaptation process. No less relevant is the choice of adaptation technique itself. Among the most popular methods one can distinguish:

- r-refinement - relocation of nodes within a computational domain toward the regions with highest errors. The mesh movement can be carried out using e.g. various deformation techniques like a linear elasticity model with an adaptation sensor as a forcing term [98, 38], or a range of error equidistribution methods [45]. The r-refinement can provide a limited reduction in error as the number of degrees of freedom is kept fixed.

- h-refinement [46, 31] - subdivision of original mesh cells based on the adaptation sensor. It is one of the most common techniques used for mesh adaptation. To make the h-refinement most effective, the treatment of hanging nodes [142] have to be implemented in the CFD solver. This allows the formation of buffer layers between the original and subdivided cells to be avoided.

- p-refinement - adjusting the order of interpolation polynomial (used for higher-order methods). A mix of hp-refinement is often used [47, 48].

- Re-meshing [43, 49] - rebuilding the original mesh based on the adaptation sensor either as a sizing function or a more general metric.

In this work, the re-meshing methodology is used because it allows exploiting a well-developed and robust meshing tools and libraries that can consistently generate high quality computational grids with a good control over the refinement and coarsening regions if the mesh.

5.2 Error estimation

5.2.1 Introduction

The discretisation error δU_h is defined as the difference between the exact solution U of the continuous PDE system $R(U) = 0$ and its discrete approximation U_h, where the latter results from the discrete PDE system solve $R_h(U_h) = 0$.

$$\delta U_h = U - U_h \tag{5.1}$$

The discretisation of system of flow equations $R(U)$ can be presented as shown in Eq. 5.2, where R_h represents a discretised system of flow equations, and $\widetilde{TE_h}$ or δR_h the remaining error due to the discrete approximation.

$$R(U) = R_h(U) + \widetilde{TE_h} \tag{5.2}$$

After re-arranging Eq. 5.2, the truncation error $\widetilde{TE_h}$, which is difference between mathematical model (PDE, $R(U) = 0$) and its discrete approximation $R_h(U)$, can be written as shown in Eq. 5.3.

$$\widetilde{TE_h} = -R_h(U) \tag{5.3}$$

It is also interesting to show how truncation errors are related to discretisation errors. This presentation can be made by first introducing Taylor expansion for a discrete residual function R_h around the exact solution U with perturbation δU_h as in Eq. 5.4.

$$R_h(U + \delta U_h) = R_h(U) + \left(\frac{\partial R}{\partial U}\right)_h \delta U_h + \dots \tag{5.4}$$

Next, disregarding higher order terms of the Taylor series, Eq. 5.5 is obtained.

$$R_h(U + \delta U_h) \approx R_h(U) + \left(\frac{\partial R}{\partial U}\right)_h \delta U_h \tag{5.5}$$

Substituting $R_h(U)$ from Eq. 5.3 using its approximation obtained based on

Eq. 5.5 leads to Eq. 5.6.

$$\widetilde{TE}_h \approx -R_h(U + \delta U_h) + \left(\frac{\partial R}{\partial U}\right)_h \delta U_h \tag{5.6}$$

However, as it was already explained, the system of flow equation $R(U) = 0$, equally the residual of the discrete solution $R_h(U + \delta U_h)$ or simply $R_h(U_h)$ is also zero for a converged solution. Hence the relation of truncation error and discretisation error can be presented as shown in Eq. 5.7 or Eq. 5.8.

$$\widetilde{TE}_h \approx \left(\frac{\partial R}{\partial U}\right)_h \delta U_h \tag{5.7}$$

$$\delta U_h \approx \left(\frac{\partial R}{\partial U}\right)_h^{-1} \widetilde{TE}_h \tag{5.8}$$

An output error δL_h due to the inexact solution can be derived in a similar way by first starting with Taylor series expansion of similarly expanded using the Taylor series:

$$L_h(U) = L_h(U_h) + \left(\frac{\partial L}{\partial U}\right)_h \delta U_h + \ldots \tag{5.9}$$

Replacing δU_h in Eq. 5.9 with the discretisation error derived from Eq. 5.8 and skipping higher-order terms, one can arrive at Eq. 5.10. Note that δR_h is an alternative notation used for truncation error $\widetilde{TE}_h$.

$$\delta L_h \approx \left(\frac{\partial L}{\partial U}\right)_h \left(\frac{\partial R}{\partial U}\right)_h^{-1} \delta R_h = \psi_h^T \, \delta R_h \tag{5.10}$$

The adjoint variable ψ for the objective function L (e.g. lift, drag) translates the truncation error δR into the error in cost function δL. In this manner, the information on how the truncation errors in each control volume contribute to the error in the cost function is obtained. Summation of all contributions together gives a scalar variable i.e. the output correction, which can increase cost function estimation accuracy. The adjoint variable ψ_h comes from the solution of the discrete adjoint system derived using Algorithmic Differentiation with Tapenade[25] (see Section 3.5). The adjoint system is presented in Eq. 5.11. Using the discrete

[25]AD tool developed at Inria http://www-sop.inria.fr/tropics/

adjoint allows exact gradients to be obtained that correspond to the cost function evaluated on the discrete space L_h.

$$\left(\frac{\partial R}{\partial U}\right)_h \psi_h^T = \left(\frac{\partial L}{\partial U}\right)_h \tag{5.11}$$

Assuming that an adjoint solution is available, the remaining task is to get a good estimate of the truncation error.

5.2.2 Truncation error estimation using geometric multigrid

The truncation error $\widetilde{TE}_h$ is the difference between a mathematical model (PDE, denoted $R(U) = 0$) and its discrete approximation $R_h(U)$, or in other words, it is the error due to the truncation of the continuous model [84, 143]. Truncation error can be calculated as shown in Eq. 5.2 and Eq. 5.3 when an exact solution U is known. First, the exact solution U has to be evaluated in the discrete space h, and then a discrete residual operator R_h has to be applied. A mathematically more rigorous way of presenting Eq. 5.3 is shown in Eq. 5.12, where an interpolation operator $\mathcal{I}^h$ is used to obtain an exact solution U in the discrete space h.

$$\widetilde{TE}_h = -R_h(\mathcal{I}^h U) \tag{5.12}$$

However, in practical applications an exact solution U is not known. Hence a method that allows estimating truncation error is required. The goal of this work is to estimate truncation error using geometric multigrid meshes, where lowercase subscript h denotes a fine discrete space, and uppercase H a coarse discrete space. Through the analogy to Eq. 5.12, a truncation error between the discrete approximations in spaces h and H can be calculated as shown in Eq. 5.13.

$$TE_H = -R_H(\mathcal{I}_h^H U_h) \tag{5.13}$$

A truncation error calculated in a discrete space H using Eq. 5.12 (with h replaced by H) and denoted $\widetilde{TE}_H$ is referred to as an exact truncation error, and it is applicable only when the exact solution U is known. A truncation error cal-

culated using Eq. 5.13 and denoted TE_H is referred to as an estimated truncation error, and it is an estimate of the exact truncation error $\widetilde{TE}_H$.

If solution in the fine discrete space h is not converged a correction can be made using Eq. 5.14, where U_h^{approx} denotes the approximate solution.

$$TE_H = -R_H(\mathcal{I}_h^H U_h) + R_h(U_h^{approx}) \tag{5.14}$$

So far, the notation $R(U)$ was used to refer to the differential form of the system of flow equations (PDE). However, STAMPS uses finite volume discretisation schemes where residual is an integral quantity over each control volume. Hence, the division by volume is required. In this work, TE_H^Ω is used to indicate the undivided truncation error in finite-volume scheme (see Eq. 5.15), whereas TE_H (Eq. 5.16) denotes the truncation error in PDE - as per its definition [62].

$$TE_H^\Omega = -R_H(\mathcal{I}^H U) \tag{5.15}$$

$$TE_H = TE_H^\Omega / \Omega_H \tag{5.16}$$

The methodology used in this work involves simplifications that can lead to errors in the estimated TE. These are as follows:

- Finite precision interpolation operators $\mathcal{I}$ are used to transfer results between fine and coarse discrete spaces. In this work, 2^{nd}-order accurate interpolation operators are used.
- Truncation error between fine and coarse discrete space TE_H is used as an approximation of the exact truncation error $\widetilde{TE}_h$. This assumption should be valid as long as meshes from both discrete spaces (h and H) lie within the asymptotic convergence range and the accuracy of interpolation operators is greater or equal to the design accuracy of the solver scheme.

As mentioned in the last point, both meshes (coarse H, and fine h), should lie within the asymptotic convergence range of solution U to get a good error estimate. However, in practice even for the cases that do not meet this criterion,

the described adaptation sensor can be successfully used to drive the adaptation process as was shown by Fidkowski [31], or as it will be shown in the results of this work.

Within the context of geometric multigrid solvers, inter-grid transfer operators between fine and coarse grids are available in the solver (Section 2.5), and the error estimation can be implemented with a little effort. The complete procedure used for truncation error estimation is presented in Algorithm 6. The final truncation error estimate TE_h is evaluated in the fine discrete space h, and it is then used to calculate an adaptation sensor.

Algorithm 6 Truncation Error Estimation Using Geometric Multigrid

1: Solve discrete system for mesh (h)	U_h		
2: Restrict solution (h) $\rightarrow$ (H)	U_H^h	$\leftarrow$	$\mathcal{I}_h^H U_h$
3: Estimate TE on the coarse grid (H)	TE_H	$\leftarrow$	$- R_H(U_H^h)/\Omega_H$
4: Prolong TE_H to fine grid (h)	TE_h	$\leftarrow$	$\mathcal{I}_H^h TE_H$

A clarification is required to justify the use of interpolation operator on truncation error. The finite volume residual (e.g. $R_H(U_H^h)$) is a dual object and a special operator with conservation properties is required to project it onto the fine mesh. However, by applying appropriate transformation, which in this case is division by volume, allows the same interpolation operator that is used for solution interpolation to be used for truncation error interpolation (defined as divided residual).

5.2.3 Output error estimation

The output error can be estimated using undivided truncation error (Section 5.2.2) and the discrete adjoint variable resulting from Eq. 5.11. The corresponding vector entries for each computational node $i = (1...N_{DoF})$ and each equation $j = (1...N_{eq})$ are multiplied element-wise as shown in Eq. 5.17.

$$OE_{h,i,j} = \psi_{h,i,j} TE^{\Omega}_{h,i,j} \tag{5.17}$$

The resulting output error field shows how the local residual error in each equation and control volume (error source) contributes to the error in the objective function ($\delta L_{h,i,j}$ / $OE_{h,i,j}$). Graphically it can be presented as shown in Figure 5.1, where the objective function is an integral over the aerofoil perimeter. The objective function L_h can be corrected as shown in Eq. 5.18.

$$L_h \leftarrow L_h + \sum_{i,j} \delta L_{h,i,j} \tag{5.18}$$

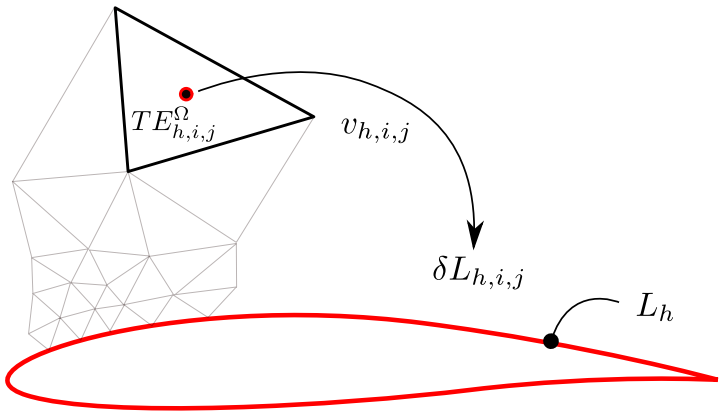


Figure 5.1: Output error ($\delta L_{h,i,j}$ / $OE_{h,i,j}$) at finite volume

5.3 Adaptation sensors

Two adaptation sensors are implemented in the two codes considered in this study (Hydra, STAMPS): a) a truncation-error-based sensor TS and b) an output-error-based indicator OS. The truncation error sensor TS is a sum of the absolute values of the truncation error from each equation as shown in Eq. 5.19. The sensor is multiplied with the control volume $\Omega^{1/3}$, which plays the role of a scaling factor that prevents infinite refinement in the regions where the errors decrease at a very low rate or diverge (e.g. at shocks) [85]. The variable N_{eq} denotes the number of equations to solve (e.g. 6 for the RANS with Spalart-Allmaras turbulence model).

$$TS_{h,i} = \Omega_i^{1/3} \sum_{j=1}^{N_{eq}} |TE_{h,i,j}| \tag{5.19}$$

The output sensor (OS) is the sum of the undivided local truncation errors weighted by the adjoint solution as shown in Eq. 5.20. Hence, it is the absolute value of the summed output errors at each control volume.

$$OS_{h,i} = \left| \sum_{j=1}^{N_{eq}} OE_{h,i,j} \right| = \left| \sum_{j=1}^{N_{eq}} \psi_{h,i,j} \; TE^{\Omega}_{h,i,j} \right| \tag{5.20}$$

5.4 Testing

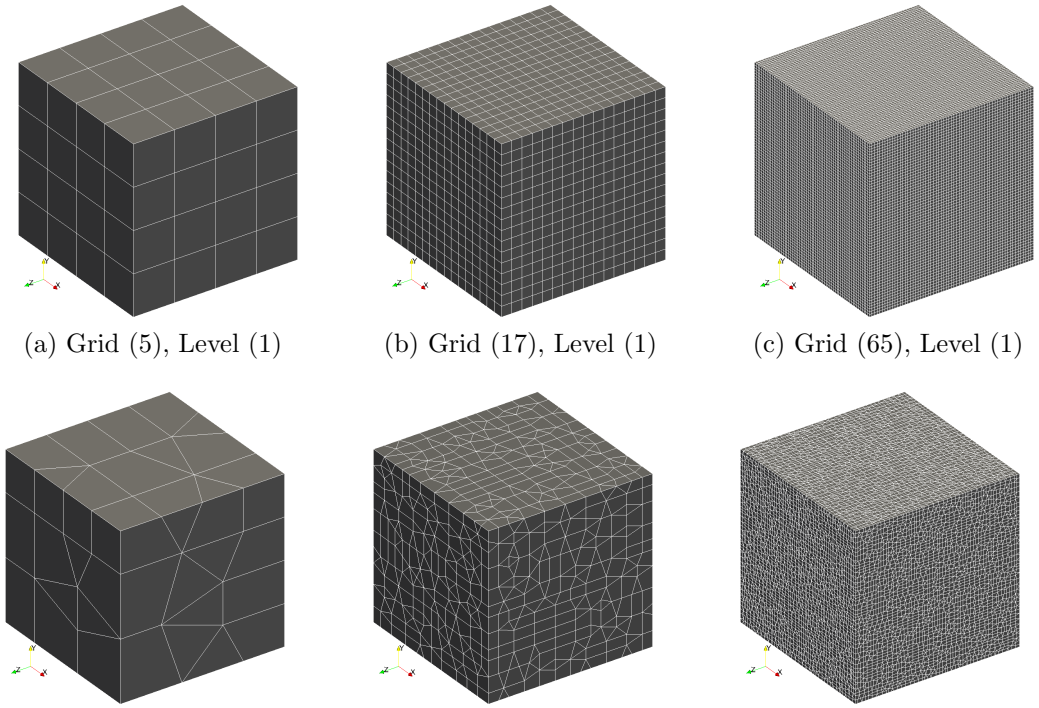
A 3D cube domain and a modified manufactured solution (see Appendix A.5) is used to perform a verification of the implemented truncation error estimation. The manufactured solution concept is a well-known strategy used in the CFD community for the verification of a solver scheme and boundary conditions implementation and was already used for STAMPS solver verification as described in Section 2.7. It is also broadly used for error analysis to verify the error estimation methods - as is the case in this work. The idea behind manufactured solution is to define a 'made up' solution using a set of continuous functions and evaluate arising source terms in the system of equations of interest - in this case, the compressible Euler equations. The derived source terms are then introduce in the discrete mesh which should converge to an exact manufactured solution as the mesh is being refined. The rate of convergence of the error in the manufactured solution allow to determine the accuracy of the solver.

When the exact solution is known the exact truncation error $\widetilde{TE_h}$ can be calculated as presented in Eq. 5.13. The error in the estimated quantity can then be calculated using Eq. 5.21.

$$\delta TE_h = \widetilde{TE_h} - TE_h \tag{5.21}$$

A set of 7 meshes was generated and used for the mesh convergence study. Mesh sizes range from 3x3x3 mesh nodes to 129x129x129, where each refinement stage was achieved by halving the coarser grid edges. An example of three refined meshes used in the study, together with corresponding coarse grids used for truncation error estimation, is presented in Figure 5.2. Note that the coarser meshes for each grid from the set were generated separately using an element-collapsing

algorithm [37]. In this manner the generated coarse grids are of a mix-cell type and are topologically not consistent with the base mesh, see the bottom row in Figure 5.2.

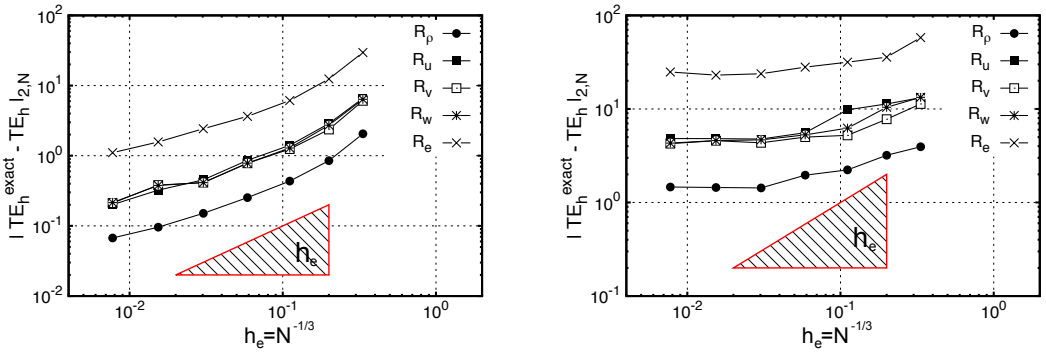


(a) Grid (5), Level (1) (b) Grid (17), Level (1) (c) Grid (65), Level (1)

(d) Grid (5), Level (2) (e) Grid (17), Level (2) (f) Grid (65), Level (2)

Figure 5.2: Set of subsequently refined meshes (top row) with corresponding coarse grids used for error estimation (bottom row)

The convergence of the L_2-norm of the error in estimated TE described by Eq. 5.21, is presented in Figure 5.3a. The error is plotted against the characteristic mesh size h_e. The convergence slope is of approximately $O(h_e)$, i.e. first-order, which is expected for the mixed cell type mesh.



(a) L_2 *norm* (b) L_∞ *norm*

Figure 5.3: Error in truncation error - mesh convergence. The variables in the legend indicate corresponding equation

Two observations should be made:

1. The slope of the converging L_2 norm of the error in TE is near $O(h_e)$ as expected for the 2^{nd}-order accurate discretisation scheme with unstructured meshes which was explained in Section 2.7.3, and shown by Lindquist and Giles [131] or Diskin [130]. Truncation error TE^{Ω} is a summation of local fluxes and thus is equal to the sum of the errors in flux reconstruction ($O(h_e^2)$) times the local area $O(h_e^2)$ divided by volume $O(h_e^3)$, which in turn gives the truncation error to be of the order $O(h_e)$ for 2^{nd}-order solution reconstruction. It is interesting to note that the truncation error for some particular cases is $O(h_e^2)$ for the edge-based solvers - see Section 2.7. This happens for two specific grid types:

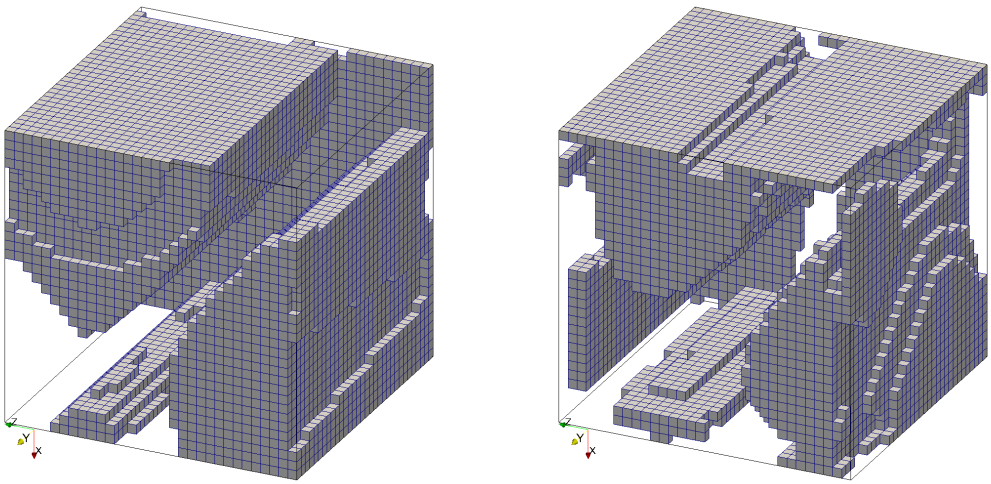
   - regular hex/quad mesh, and
   - general tetrahedral/triangular grids.

   For these two cases, the supra-convergence property is obtained due to the cancellation of the $O(h_e)$ errors which was shown in Section 2.7.3 or by Diskin [63]. As a result, the accuracy of truncation error estimation is improved.

2. When inspecting the results in Figure 5.3a it can be noticed that he slope of L_2 norm of truncation error reduces slightly with decreasing h_e. Inspecting further using the infinity norm of the error shown in Figure 5.3b, a control volume with slowest converging error is revealed. It was found that

this control volume is located at the boundary of the computational domain. Hence a reduction of accuracy for some control volumes is expected. This could have a negative effect on the refinement process as the errors at the some control volumes would not reduce while mesh is refined, thus leading to excessive adaptation. To avoid over-refinement of the cells that experience reduced order of error convergence, the truncation sensor is obtained by multiplying truncation errors with the characteristic size of cell volume ($\Omega^{1/3}$) as presented in Eq. 5.19. The output-based sensor includes the volume-scaling in its definition, see Eq. 5.20.

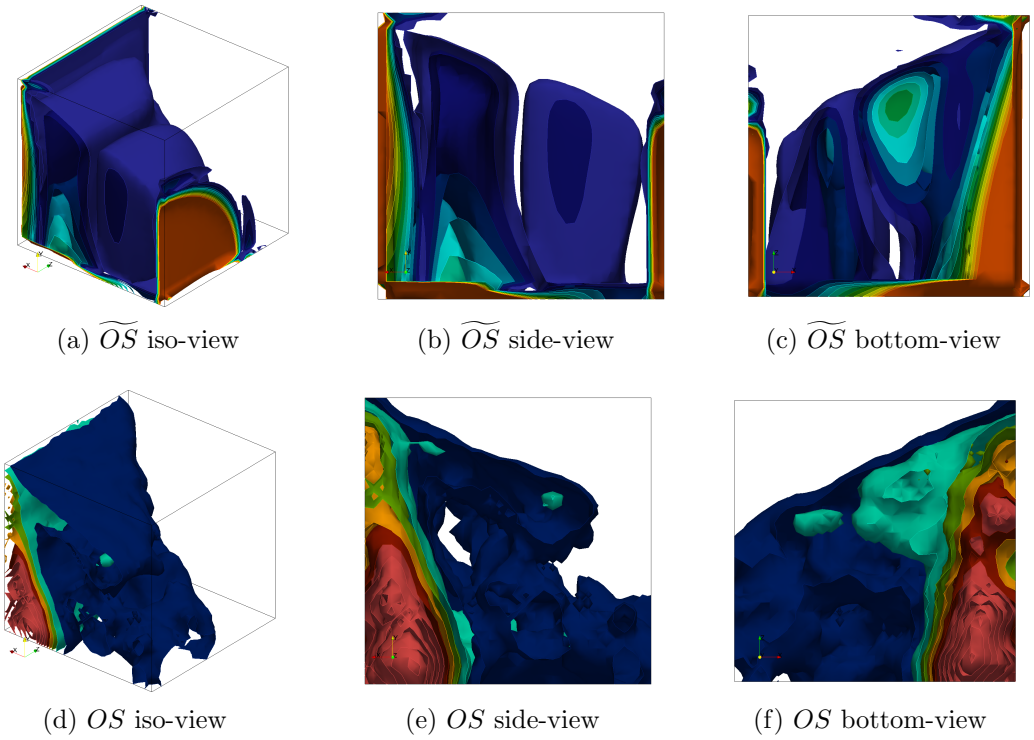
The L_2 norm of the error in TE estimation converges as the mesh is being refined. This confirms the argument from Section 5.2.2 that the estimated truncation errors on a coarse grid can be used as a representation of fine grid errors. Furthermore, the qualitative comparison of exact and estimated TE shows a very similar pattern of error distribution within the computational domain, even though the coarse grid used for truncation error estimation is topologically inconsistent with the fine mesh - Figure 5.4. The figure shows the regions of cells created with Paraview's 'Threshold' tool. The threshold was adjusted for both cases to get a roughly similar volume fill. The results confirm that the estimated errors should lead to a reasonable indication of the mesh regions marked for refinement.



(a) $\widetilde{TE}$ (b) TE

Figure 5.4: Qualitative comparison of the exact truncation error ($\widetilde{TE}$) and the estimated truncation error (TE) for grid (33), continuity equation

Figure 5.5 shows a qualitative comparison of the final output sensor as defined in Eq. 5.20, for the exact and estimated truncation errors used. As the objective function for the adjoint solver the drag evaluated on one of the cube sides (see Figure 5.8a) was used. The general pattern of error distribution within the domain shows a good match, even though the coarse grid used for (TE) estimation was topologically inconsistent with the fine mesh.



(a) $\widetilde{OS}$ iso-view (b) $\widetilde{OS}$ side-view (c) $\widetilde{OS}$ bottom-view

(d) OS iso-view (e) OS side-view (f) OS bottom-view

Figure 5.5: Output sensor based on exact truncation error - $\widetilde{TE}$, (top row) vs. output sensor based on estimated truncation error - TE, (bottom row)

The key conclusion from the testing stage is that the presented error estimation methodology with the topologically inconsistent coarse grid is expected to be sufficient for correct indication of refinement areas within the computational domain. This argument is refined in Section 5.5.

5.5 Re-meshing refinement with BoxerMesh

Rolls-Royce's proprietary code Hydra was used to obtain results for this section. Hydra is a compressible RANS flow and adjoint solver that exploits a vertex-centred (edge-based) finite-volume discretisation scheme, nominally 2^{nd}-

order accurate on a general mesh type - verified in this work by using a method of manufactured solution. The main solver of Hydra is very similar to STAMPS iterative scheme, and both codes use a geometric multigrid technique, which makes it convenient for the implementation of truncation error estimation as described in Section 5.2.2.

The proposed output-based sensor estimation methodology (Section 5.2) was implemented and tested in Hydra. A small number of explicit smoothing iterations (see Appendix A.7) was applied to the obtained refinement indicator to regularise unwanted high-frequency modes arising mainly from the topological inconsistency between meshes used for truncation error estimation. The obtained sensor fields can either be used for hierarchical refinement, leading to topologically consistent grids, or be used as local sizing fields in a re-meshing procedure as is the case in this work. The obtained fields are used to drive a re-meshing process using BoxerMesh[26] [86]. The application of the output-based re-meshing refinement to the simple cube case with the inviscid flow is presented in Section 5.5.2.

5.5.1 Procedure description

The re-meshing approach using BoxerMesh and the output-based sensor defined in Eq. 5.20 is used to drive the adaptation process. The mesher uses an octree cut-cell algorithm to create an initial mesh respecting defined areas of refinement. This mesh is then fitted to the geometry defined by the user and in the final step the boundary layer is extruded. An example cross-section of the stator blade mesh is presented in Figure 5.6.

[26]http://www.cambridgeflowsolutions.com/en/products/boxer-mesh/

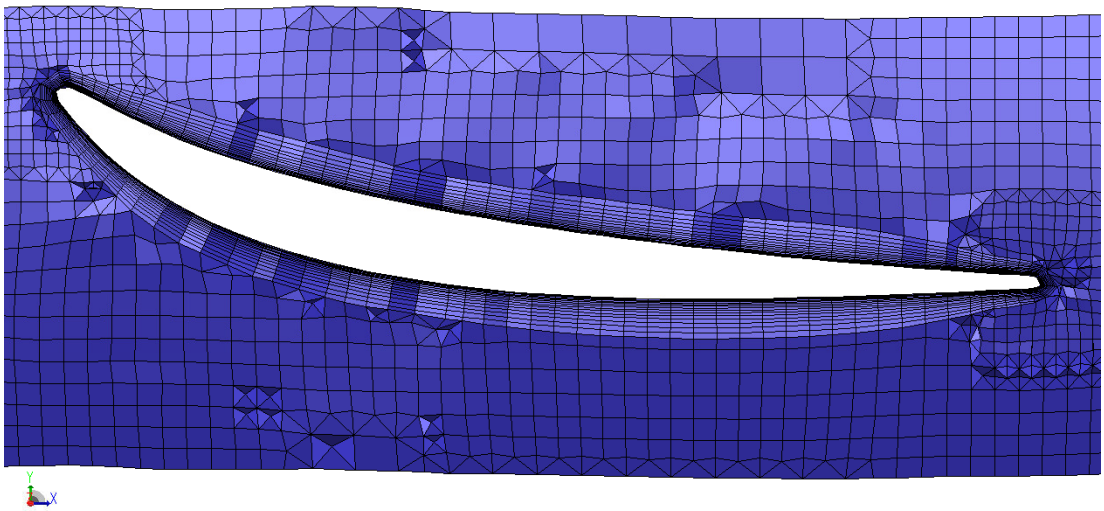


Figure 5.6: Cross-section of the stator mesh generated in Boxer

The procedure for a single re-meshing step is as follows:

1. Obtain the flow solution (U_h).
2. Estimate the truncation error (TE_h) as presented in Algorithm 6.
3. Obtain the adjoint solution (ψ_h).
4. Evaluate the output-based sensor (OS_h) - Eq. 5.20.
5. Perform 5 explicit smoothing iterations (see Appendix A.7) on the obtained sensor (OS_h) to damp the unwanted high-frequency modes.
6. Use Paraview to extract the mesh region for refinement.
    - Use the 'Threshold' option to mark the region for refinement, Figure 5.7.
    - Extract the iso-surface and output an STL file.
7. Import the iso-surface to Boxer and specify the new refinement region for the octree mesher.
8. Generate the new mesh and re-run the case.

In the current work, steps (5-8) required manual operations from the user.

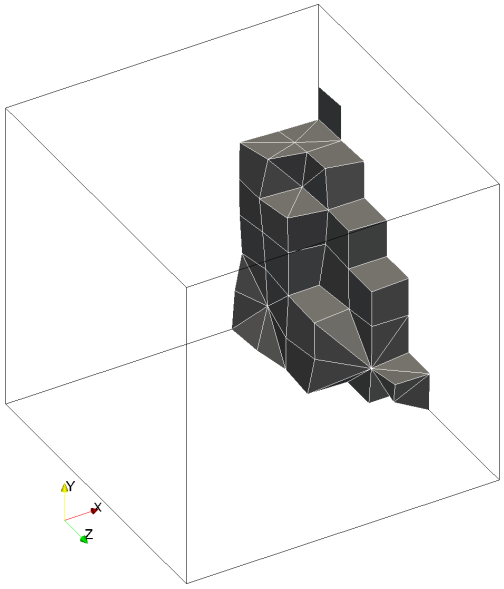
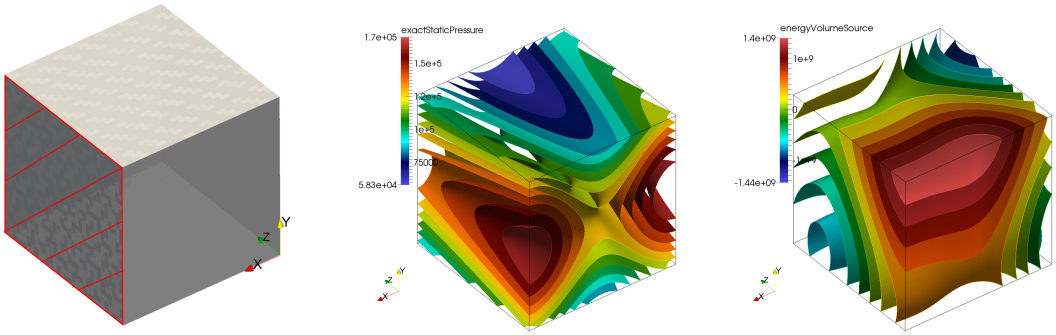


Figure 5.7: Cells forming region for refinement obtained using 'Threshold' feature in Paraview

5.5.2 Cube with 3D manufactured solution

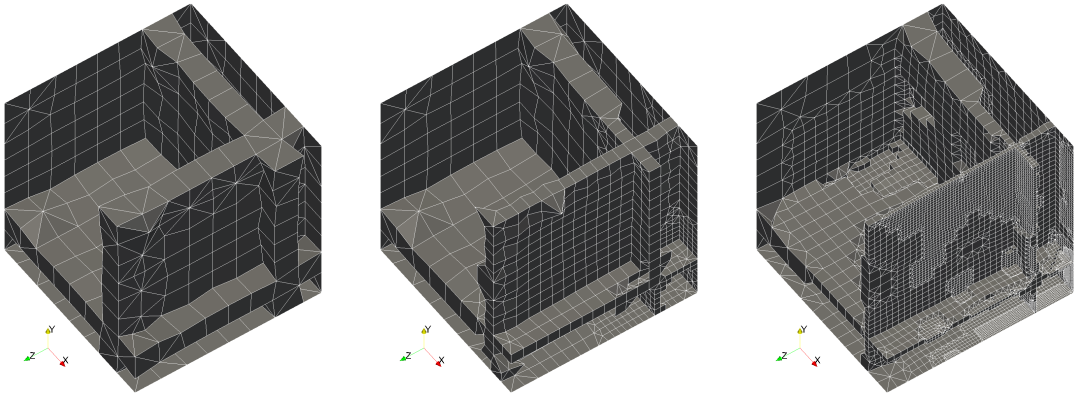
As the first output-based re-meshing example the cube case with 3D manufactured solution [1] is used. Although the case is physically meaningless, it is challenging for the solver as it uses a highly nonlinear set of functions (mix of sines and cosines). It is a compressible, supersonic Euler flow where the example pressure field and corresponding manufactured source term are presented as in Figure 5.8. The objective function is a drag force integrated over the side of the cube marked in Figure 5.8a.



(a) Patch for objective evaluation (b) Pressure (p) (c) Energy src (f_e)

Figure 5.8: The 3D supersonic manufactured solution

The initial grid was generated using Boxer and is of mixed cell type. The coarse grid for truncation error estimation was generated using an internal Rolls-Royce element-collapsing tool. The re-meshing was performed according to the procedure described in Section 5.5.1. The total two re-meshing steps were applied, and the resulting refined meshes are presented in Figures 5.9b and 5.9c. The complex and non-intuitive refinement structures are clearly visible. Unfortunately, It is hard to connect the resulting refinement structures in the mesh to any specific flow phenomena as the used manufactured solution doesn't have any physical meaning. Mesh refinement on a physically meaningful more case is presented in Section 5.6.



(a) Initial mesh (b) Re-mesh step 1 (c) Re-mesh step 2

Figure 5.9: Re-meshing process driven by the output-error-based sensor

Figure 5.10 shows the comparison of the achieved error in the objective function for the uniformly refined set of grids - 3x3x3 to 129x129x129, regular hex where each refinement stage was achieved by halving the coarser grid edges - and the output-based re-meshed grids. The latter approach achieves an improved convergence slope. The error in objective function estimate for the output-based adapted mesh decreases at the rate between $O(h_e^3)$ and $O(h_e^4)$, whereas for the uniformly refined grid the errors decrease at the rate of $O(h_e^2)$.

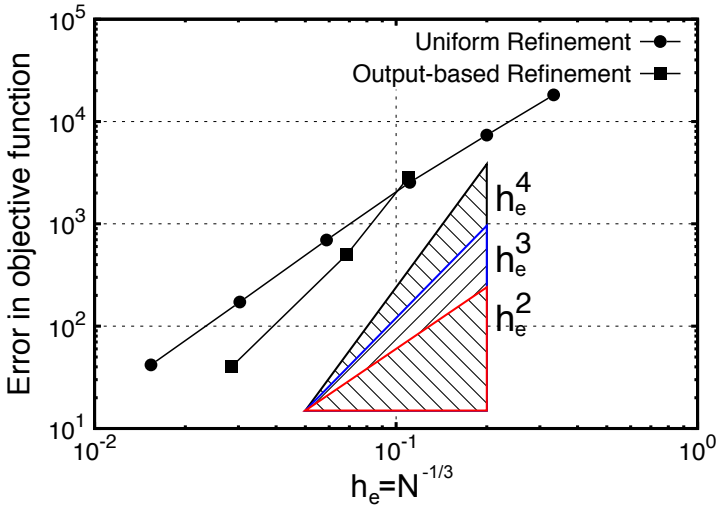


Figure 5.10: Re-meshed vs uniformly refined regular mesh - error convergence comparison

Using results from Figure 5.10 the node count of a uniformly refined grids can be estimated that achieves similar error level as the output-based adapted levels 0, 1, and 2. The results are shown in Table 5.1. For the mesh from adaptation step 2 (Figure 5.9c) almost an order of magnitude lower node count was achieved as compared to an estimated size of a uniformly refined mesh, even though a rather crude re-meshing methodology was used.

Re-meshing stage	$\delta L/\widetilde{L}$ [%]	N^{OS}_{DoF}	N^{U}_{DoF}	N^{OS}_{DoF} / N^{U}_{DoF}
0	2.11	754	660	0.87
1	0.37	3082	12100	3.9
2	0.03	43349	335000	7.7

Table 5.1: Number of degrees of freedom of adapted mesh and achieved error level vs. corresponding estimated (interpolated) node count of a uniformly refined regular hex meshes required to achieve the similar error values. DoF - degrees of freedom, N^{OS} - DoF for output-based refinement, N^{U} - DoF for corresponding uniformly refined grid

The results shows that the truncation error estimation method that uses a coarse geometric multigrid mesh (which in a general case is topologically inconsistent) can provide a good error estimate as long as the accuracy of interpolation operators is consistent with the nominal order of accuracy of the discretisation scheme. Using the estimated truncation error and weighting it using adjoint solu-

tion provides a robust output-based adaptation sensor that improves convergence rate of the error in objective function as compared to uniformly refined grids.

The proposed methodology for truncation error estimation is more general than the truncation error estimation method used by Ponsin and Fraysse [34] as well as Venditti or Fidkowski [32, 31] which are based on nested grids. Furthermore, the proposed methodology uses coarse mesh for error estimation which is computationally less intensive than using a nested fine grid in the work by Venditti or Fidkowski [32, 31] .

5.6 Re-meshing refinement with mmg3d

In this section, another re-meshing refinement example is presented using inviscid Onera M6 wing. First, the refinement procedure is described where the in-house code STAMPS is used to solve flow and adjoint equations as well as to provide an adaptation indicator. The mmg3d tool is used to adapt mesh using a re-meshing technique. An output-based adaptation sensor is used to determine the sizing field used for re-meshing. The results are presented for two flow conditions: a) subsonic Ma= 0.1 (Section 5.6.3), and b) transonic Ma= 0.84 (Section 5.6.4). In both cases, a comparison between output-based adapted and uniformly refined grids is presented.

5.6.1 Introduction

In Section 5.5, a semi-manual adaptation technique was used using BoxerMesh. First, the iso-surface that encloses volume with highest error value was generated, then the meshing tool was instructed to halve the cell size within the region inside the iso-surface. In this section a different tool called mmg3d is used that allows the adaptation process to be fully automated. Mmg3d is a robust, open-source and multi-disciplinary software for re-meshing (developed by Mmg Open Source Consortium). It can generate and manipulate tetrahedral meshes and can be easily coupled with custom applications. The input to mmg3d is a scaling factor field that adapts a mesh relative to the current cell sizes. For example, if a scaling factor of 0.5 is assigned at a specific mesh cell, the algorithm will try to

halve the original edge length of this cell. Mmg3d capabilities cover isotropic and anisotropic mesh generation. In this work only the isotropic re-meshing functionality was used.

The procedure of re-meshing adaptation with mmg3d is presented in Figure 5.11.

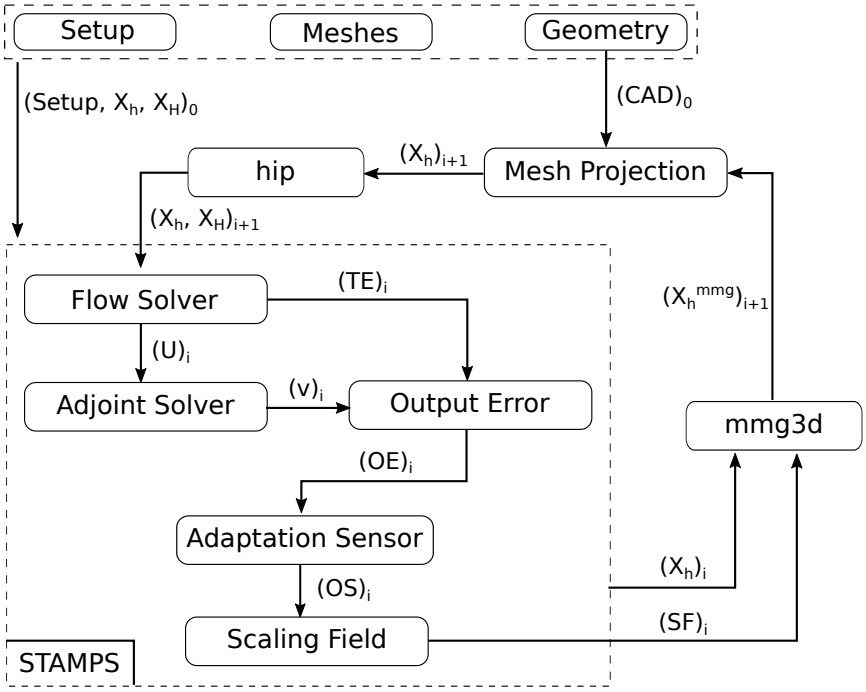


Figure 5.11: Mesh adaptation flowchart

As a first step, the initial CAD geometry is prepared, and an initial set of geometric multigrid meshes generated using a hip tool [37], which has already been introduced in Sections 2.5 and 2.6. The setup and all pre-processing operations are enclosed in the dashed box in the top part of Figure 5.11. After the initialisation step, all the information (setup, meshes) is provided to the STAMPS solver, and a flow solution is obtained. Next, the truncation error is estimated and weighted using an adjoint variable that results from the adjoint system solve; then the output error is calculated. Error estimation is carried out according to the procedures already described in Section 5.2. The final step executed by STAMPS is generation of the scaling factor field. The scaling factor is obtained based on the values of the adaptation sensor. In this work the scaling factor was set to vary between values of 0.5 and 2, where the lower bound corresponds to

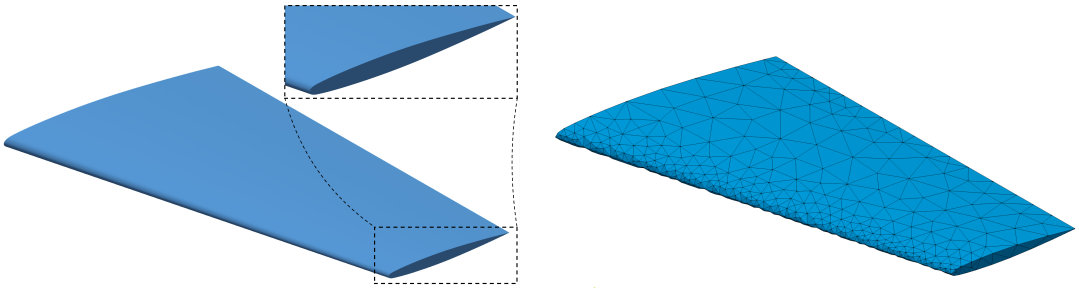
the halving edge length of a cell (refinement), and the upper bound to doubling it (coarsening). The coarsening procured is also limited to the maximum edge length corresponding to 0.25 of the largest dimension in the computational domain to prevent over-coarsening, and maintain the original domain bounds. In the current work, 20% of all cells are marked for refinement and 20% for coarsening. This choice is to adapt or coarsen only a smaller fraction of all cells with highest and lowest error values respectively and limit the amount of refinement and coarsening within a single adaptation step. A similar approach was proposed by Fidkowski et al. [81]. As the mesh is being refined, the solution to the nonlinear flow equations evolves and the highest error concentration regions of computational domain may change (e.g. formation of a shock wave). Furthermore, the accuracy of error estimation increases as the mesh is adapted. In summary, the change in cell sizes is limited between factor 0.5 and 2, 20% of cells are refined and 20% coarsened. The process is performed in an iterative fashion.

After the scaling field is obtained, mmg3d is executed and the re-meshing refinement/coarsening is performed based on the original mesh and scaling factor field. Next, the resulting mesh is projected onto the original CAD geometry in order to preserve the shape of the M6 wing. Finally, the hip tool is used to generate coarse geometric multigrid mesh. The refinement process is continued until interrupted by the user. In future work, more refined stop criteria for refinement could be introduced such as maximum number of degrees of freedom or minimum edge size.

5.6.2 M6 wing reference meshes and setup details

The Onera M6 wing was introduced in Section 3.7. Figure 5.12 shows CAD geometry of the wing and the initial mesh. The mesh was made deliberately coarse to start the adaptation process with small mesh and adapt iteratively only where the sensor values are high or coarsen further if the sensor values are low. As can be seen in Figure 5.12a, the initial coarse mesh is not resolving geometry well, especially near the leading edge. For this reason, after every adaptation iteration a mesh projection step (see Figure 5.11) is required. To simplify mesh projection the original M6 wing CAD geometry was modified by making a sharp cut near

the tip (see Figure 5.12a). The mesh projection step is done in the following sequence. First, the leading and trailing edge of the mesh is being deformed to its original position. Next, the z-coordinate of wing tip surface nodes is re-set to the initial value. Finally, the mesh nodes of the top and bottom wing surfaces are projected on the CAD geometry using surface normal direction.



(a) Geometry with tip cut (b) Initial mesh (4,882 nodes)

Figure 5.12: M6 wing geometry and initial mesh for adaptation

Details on the solver setup, reference meshes, objective function used for output-based adaptation, and scaling factor calculation are provided.

Setup

Figure 5.13 shows the computational domain. It consists of a wing geometry enclosed by a symmetry plane and freestream boundary.

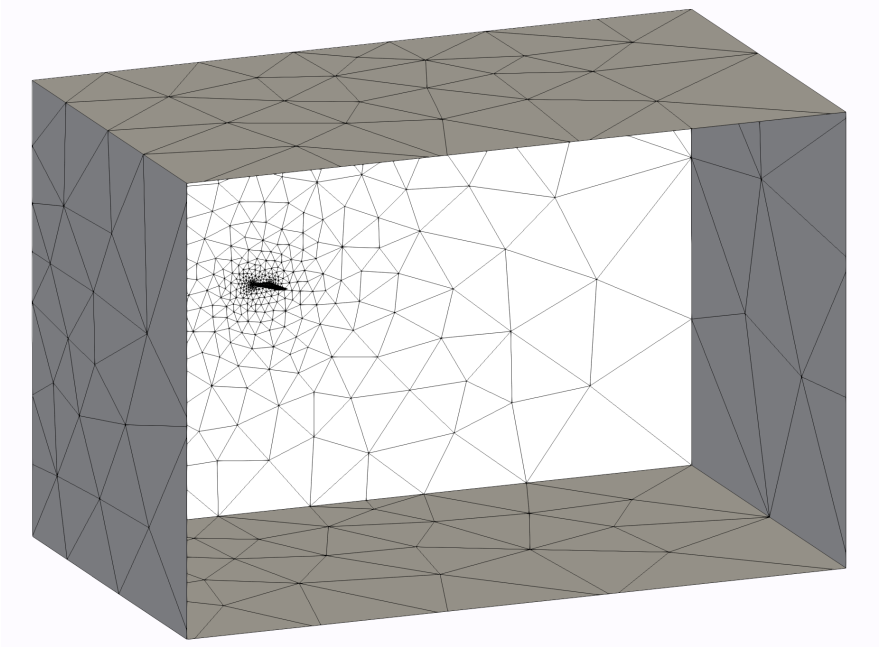
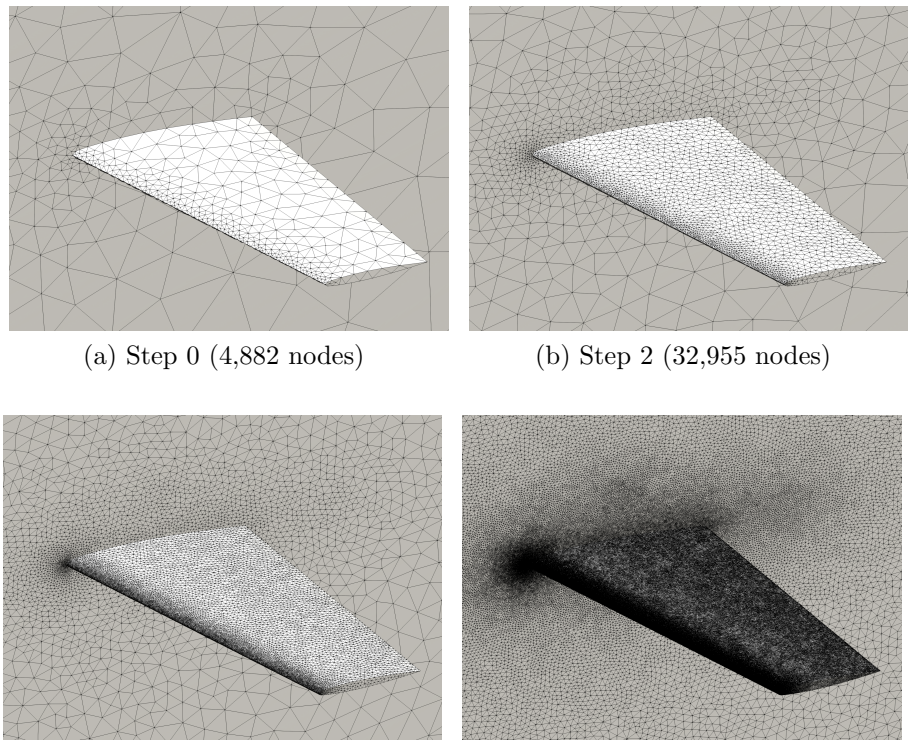


Figure 5.13: Computational domain of M6 wing (step 0)

In both adaptation examples, the angle of attack at freestream condition was set to $AoA = 3.06°$. The Mach number in the far-field was set to Ma= 0.1 for the subsonic case, and Ma= 0.84 for the transonic case. An inviscid compressible flow solver is used in both cases. For the subsonic case no slope limiter was used, and for the transonic example a default DMV limiter was exploited (see Section 2.3.6). The default settings were chosen for all other STAMPS options as presented in Section 2.1. Truncation error, output error, and the final output-based sensor were calculated according to the method described in Section 5.2.

Reference meshes

A set of uniformly refined grids was created starting from the initial mesh shown in Figure 5.14a. The coarsest mesh was created using typical engineering best practice discussed already in Section 1.3. A refinement was applied at the leading edge and wing surface and coarse mesh was set at the outer boundary of the computational domain (see Figure 5.13). Example grids for the uniformly refined set are shown in Figure 5.14.



(a) Step 0 (4,882 nodes)

(b) Step 2 (32,955 nodes)

(c) Step 3 (192,126 nodes)

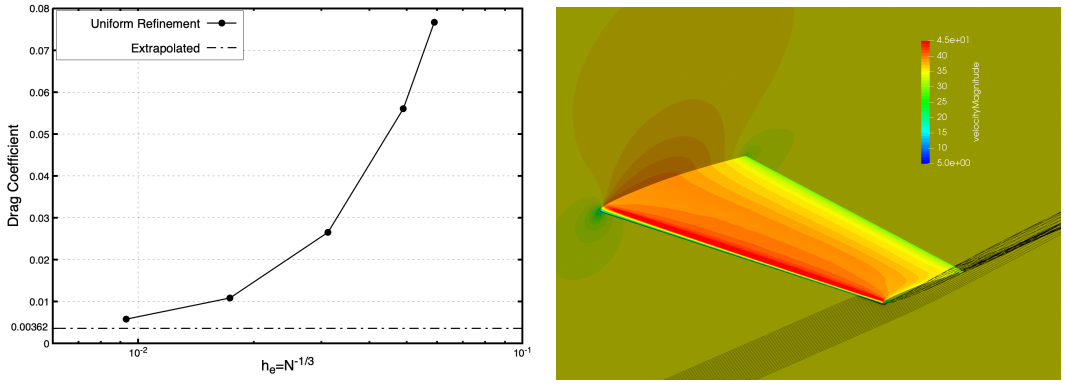
(d) Step 5 (8,359,233 nodes)

Figure 5.14: Examples of uniformly refined meshes created using typical engineering best practice

Objective function

In both adaptation examples, the objective function is drag coefficient of the wing. In order to perform an error convergence study of the functional as the mesh is being refined, an exact drag coefficient value is required. In the subsonic inviscid case the non-zero drag force is related to the induced drag due to the lift generated by the wing body for a non-zero angle of attack. As a result, a wing tip vortex is created (see Figure 5.15b) that consumes energy and results in the induced drag. Additionally, there is also a drag force related to numerical diffusion introduced in the discretisation scheme, but this component vanishes as the mesh is refined ($h_e \to 0$).

The convergence of the drag coefficient for the uniformly refined grids (introduced in Figure 5.14) and an example velocity contour plot with streamlines near the wing tip are shown in Figure 5.15. The exact drag coefficient was estimated using Richardson extrapolation [9]. The value obtained was $C_D = 0.00362$.



(a) Drag coefficient convergence for uniformly refined mesh

(b) Velocity contours and flow streamlines near the wing tip

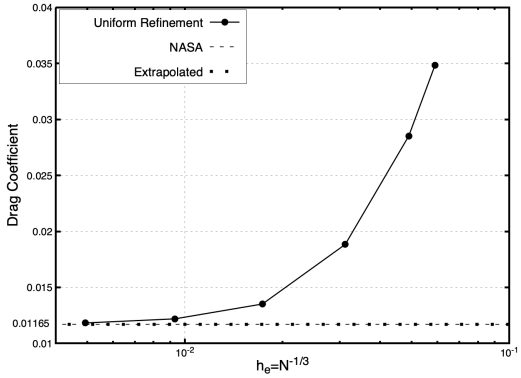
Figure 5.15: Objective function convergence and example flow visualisation for subsonic M6 wing

The exact value of drag coefficient in the transonic example, which corresponds to the sum of wave drag and induced drag, is estimated using extrapolated value from the mesh convergence study shown in Figure 5.16a. The value obtained was $C_D = 0.01165$. A comparison to the pressure drag coefficient for the M6 wing available on the NASA website[27] is also presented in Figure 5.16a ($C_D^{NASA} = 0.0117$). The difference between the extrapolated drag coefficient and NASA

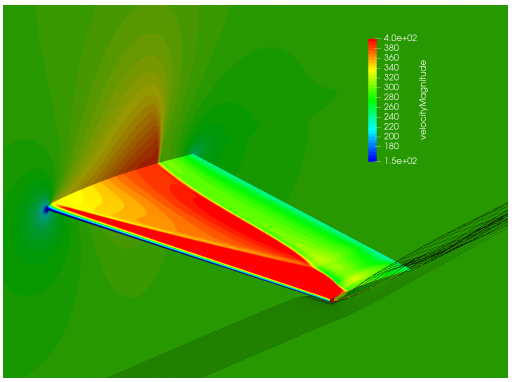
[27] https://turbmodels.larc.nasa.gov/onerawingnumerics_val_sa.html

results is related to the modification of the wing tip for the geometry used in this work (see Figure 5.12a).

An example velocity contour plot with streamlines near the wing tip for the transonic M6 wing is shown in Figure 5.16b.



(a) Drag coefficient convergence for uniformly refined mesh



(b) Velocity contours and flow streamlines near the wing tip

Figure 5.16: Objective function convergence and example flow visualisation for transonic M6 wing

Scaling factor

Mmg3d uses a scaling field to determine mesh sizing relative to the input mesh. The scaling-factor field is obtained based on an adaptation sensor, in this case the output-based sensor described in Section 5.3. First the sensor field is evaluated then sorted in ascending order. The original mesh indices are stored. Next, a scale factor between 0.5 and 1 is assigned to 20% of all control volumes with the highest sensor value, and a scale factor between 1 and 2 is assigned to 20% of control volumes with the lowest sensor value. Figure 5.17 shows error sensor distribution for the initial mesh (see Figure 5.14a) and associated scaling factor according to the described rule. A linear change of scaling factor is used.

In principle, an adaptation sensor could be used to create scaling factor according to sensor distribution. However, as the spread in value is high and increases sharply to maximum value (and decreases sharply towards minimum value) this approach could lead to very little refinement and coarsening. The linear scaling was chosen to encourage more refinement for the control volumes with the highest sensor values and coarsening for the control volumes with the lowest sensor

values, while preventing too sharp a decay of the scaling field towards 1.

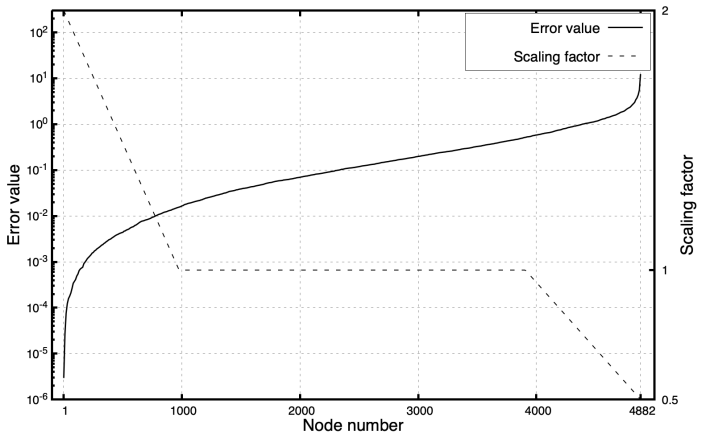


Figure 5.17: Distribution of error sensor value (sorted ascending) and scaling factor for the initial mesh

5.6.3 Subsonic M6 wing

Output-based mesh adaptation is first applied to the inviscid Onera M6 wing at subsonic conditions (Ma= 0.1, $AoA = 3.06°$). Figure 5.18 shows a comparison of drag coefficient convergence between the uniformly refined grids (see Section 5.6.2) and output-based adapted meshes.

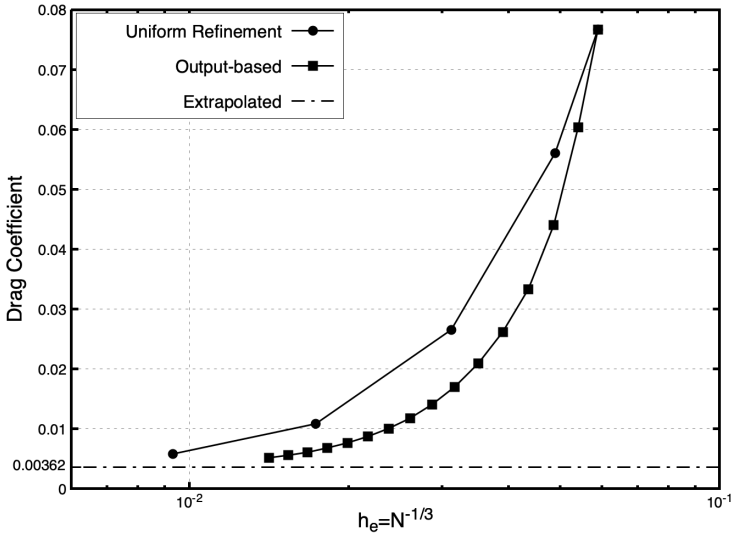


Figure 5.18: Comparison of convergence of drag coefficient between uniformly refined, output-based adapted, and output-based corrected meshes for subsonic M6 wing

The output-based adapted mesh outperforms uniformly refined meshes created according to engineering best practice.

Figure 5.19 shows the results of convergence of error in drag coefficient as the mesh is refined. Logarithmic scales are used on both axes. The 3 additional dashed lines show convergence slopes that correspond to error converging at rates between h_e and h_e^3. The error in predicted drag coefficient converges with second-order slope for the uniformly refined grids as expected for the 2^{nd}-order accurate discretisation scheme. Output-based adapted grids allow improved convergence slope of the error in drag coefficient to be achieved and converge at the slope near 3^{rd}-order.

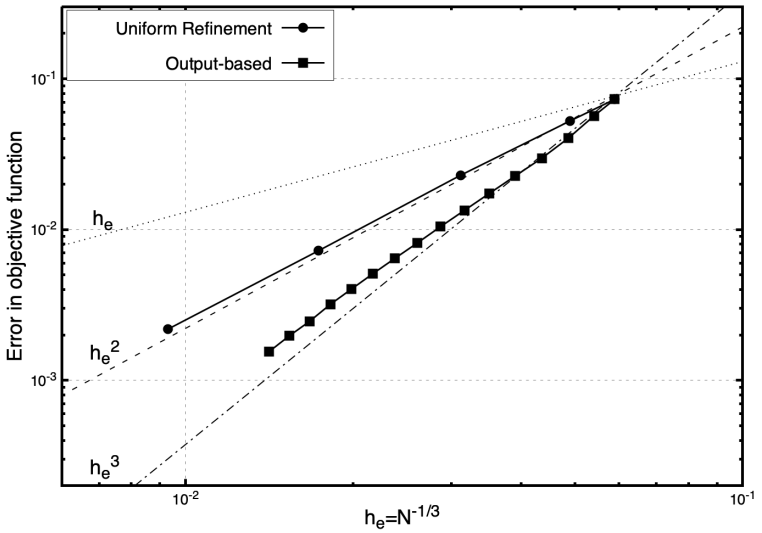
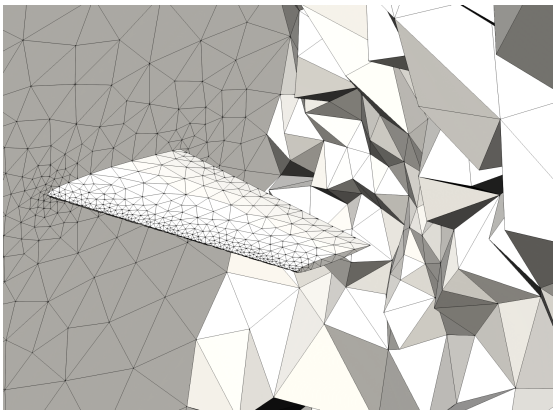
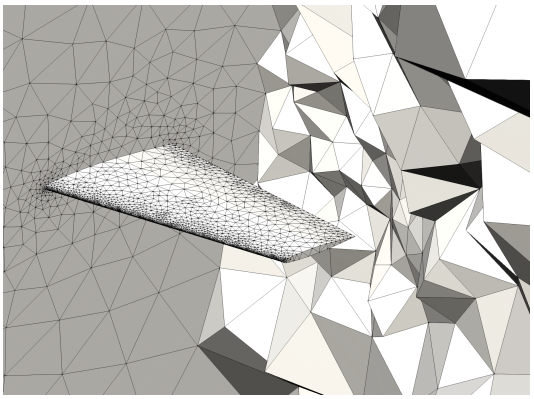


Figure 5.19: Comparison of convergence of error in drag coefficient between uniformly refined, output-based adapted, and output-based corrected meshes for subsonic M6 wing

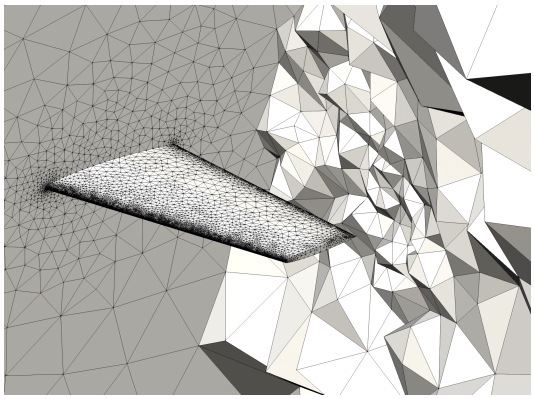
Figures 5.20 shows initial mesh and adapted steps. Each step corresponds to a point in Figure 5.19. The results highlight several key regions of the computational domain where the strongest refinement took place. First, the leading and trailing edges were strongly refined to better resolve the geometry and rapidly reduce the amount of numerical viscosity in the discretisation scheme. Next, around refinement step 8 (Figure 5.20c) adaptation started focusing around the wing tip and downstream of the wing tip while still adapting the leading and trailing edges. The refinement around the wing tip is justified because resolving wing tip vortex (see Figure 5.15b) is important for accurate prediction of induced drag.



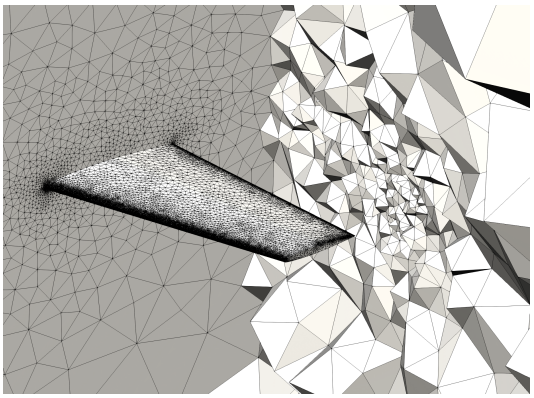
(a) Step 0 (4,882 nodes)



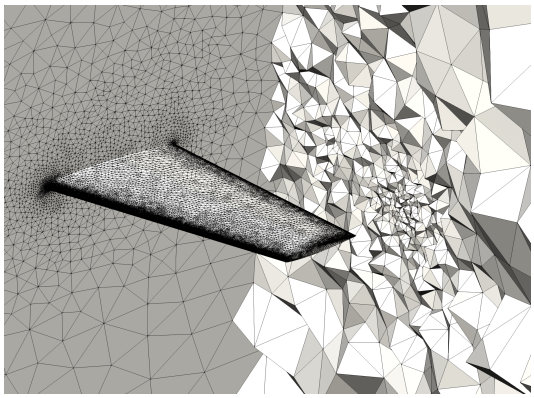
(b) Step 4 (16,844 nodes)



(c) Step 8 (56,353 nodes)



(d) Step 12 (165,500 nodes)

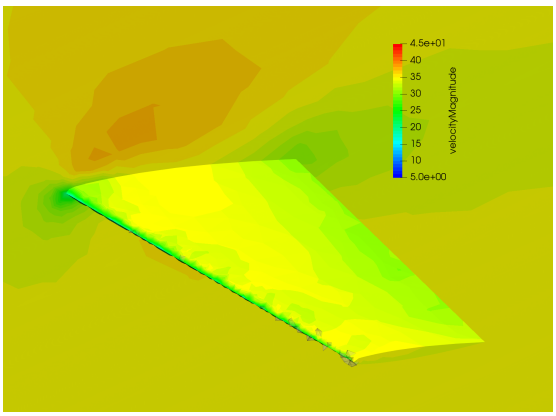


(e) Step 15 (354,006 nodes)

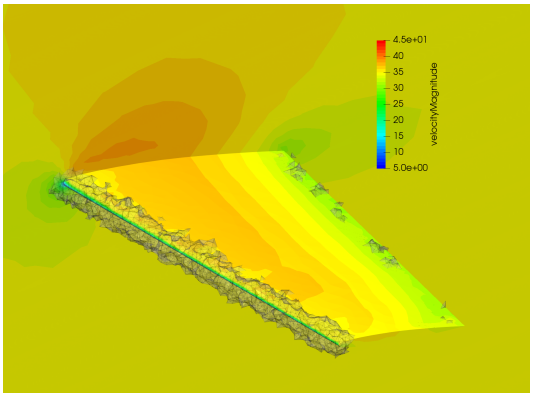
Figure 5.20: Mesh evolution for subsonic M6 wing - wing, symmetry plane, and cut-plane downstream of the wing

Figure 5.21a-5.21e shows the velocity contour plot for 5 adaptation steps. The semi-transparent volume highlight regions where the smallest cells are present. A constant threshold value was used for each mesh of 10 mm^3, which corresponds to an edge length of isotropic cell of around 2.2 mm. The initial solution for step 0 is resolving the flow very poorly with a high amount of numerical viscosity causing

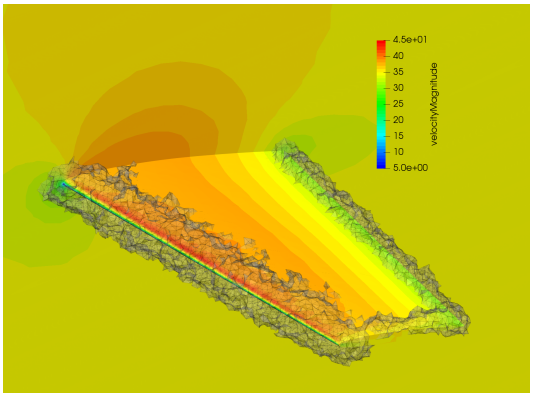
the peak velocities to be low, whereas mesh from adaptation step 15 resolves the geometry and flow accurately.



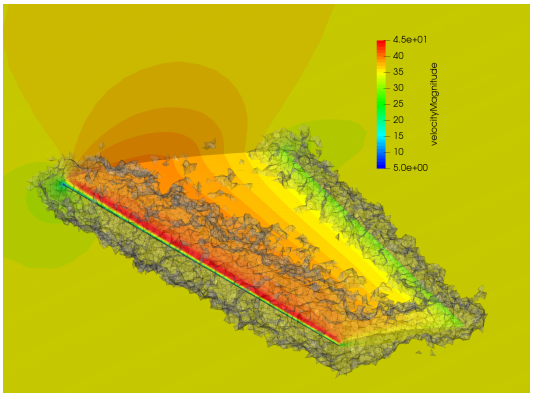
(a) Step 0



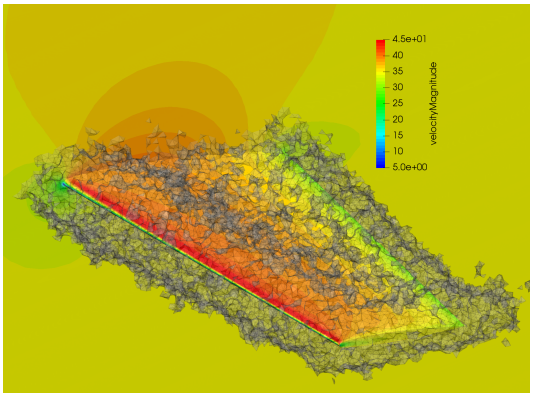
(b) Step 4



(c) Step 8



(d) Step 12

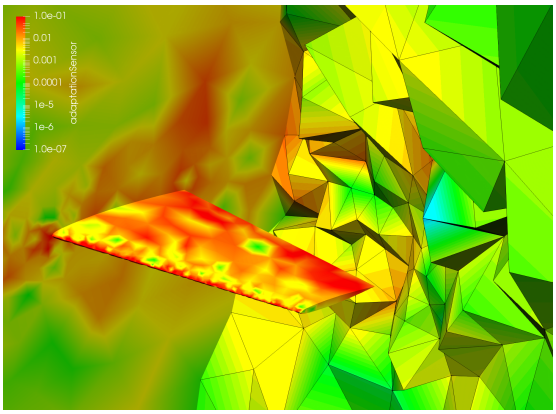


(e) Step 15

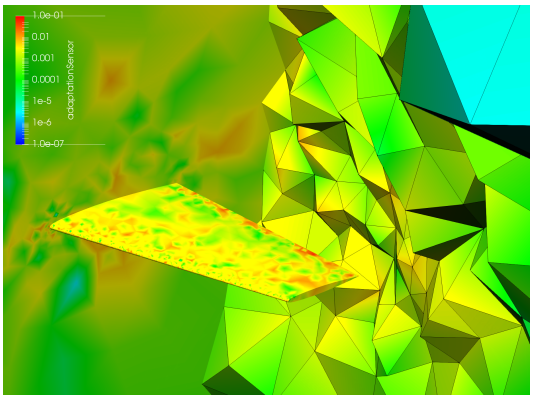
Figure 5.21: Mesh adaptation and evolving solution for subsonic M6 wing (velocity contours). Semi-transparent iso-volume marks finest cells (volume below a threshold of 10 mm^3 - cell edge length of around 2.2 mm)

Figures 5.22a-5.22e present changes in adaptation sensor value with progressing adaptation. The scale was fixed for all figures, and a logarithmic scale was used. Sensor value decreases, starting from level 0.1 on the wing surface at step

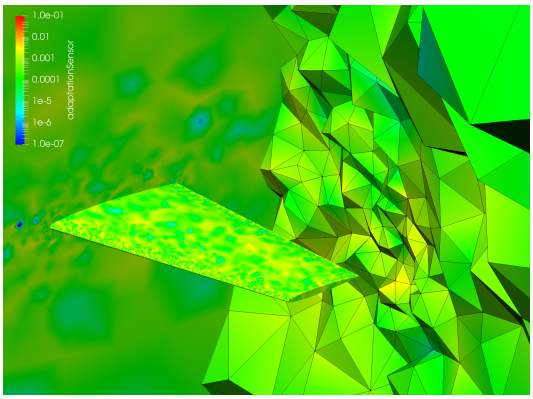
0 towards values below 10^{-4} at step 15.



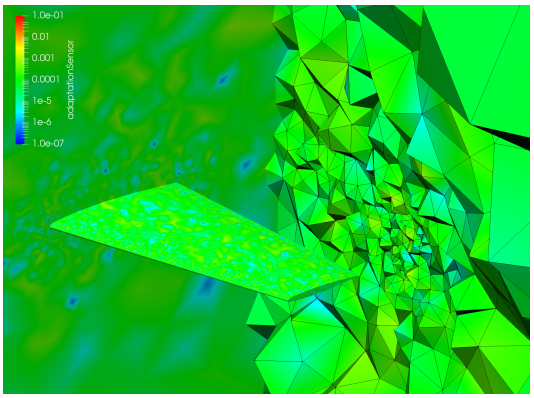
(a) Step 0



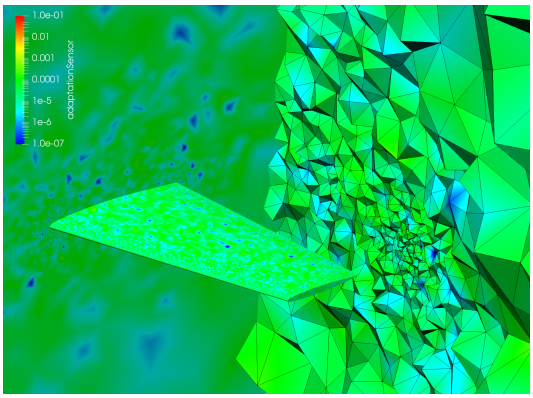
(b) Step 4



(c) Step 8



(d) Step 12



(e) Step 15

Figure 5.22: Adaptation sensor evolution for subsonic M6 wing (logarithmic scale) - wing, symmetry plane, and cut-plane downstream the wing

5.6.4 Transonic M6 wing

In this section, another application of output-based re-meshing using mmg3d is presented using the Onera M6 wing at transonic flow conditions. Figure 5.23 shows a comparison of drag coefficient convergence between the uniformly refined grids (see Section 5.6.2) and output-based adapted meshes.

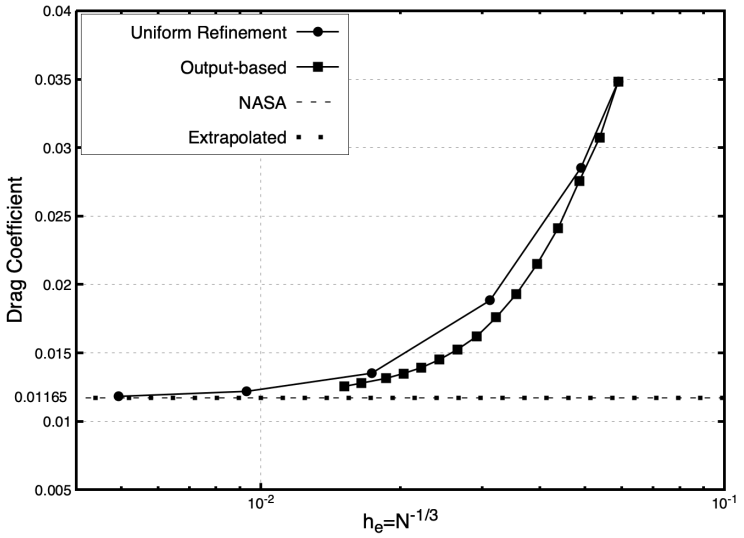


Figure 5.23: Comparison of convergence of drag coefficient between uniformly refined, output-based adapted, and output-based corrected meshes for transonic M6 wing

Analysis of convergence slope of the error in drag coefficient is presented in Figure 5.24. The results show that error decreases with a slope close to h_e^2 for uniformly refined mesh and with a slope near $h_e^{2.4}$ for the output-based adapted grid. Although the slope of converging error is improved for output-based adapted mesh, the improvement is reduced as compared to the subsonic M6 wing example shown in Section 5.6.3. There are several possible reasons of this result. First, in this case the DMV slope limiter was used which can affect accuracy of estimation. Second, the adjoint solver was not verified for the case with limiters which can be another source of worse quality of the estimate of the output sensor. Finally, the case is run at transonic conditions and strong shock waves appear on the wing which create a more challenging flow condition and may cause over-refinement in the regions near the shock lines.

Despite the above, the re-meshing refinement method shows improvement, and can be beneficial for applications with high a Mach number and shock waves.

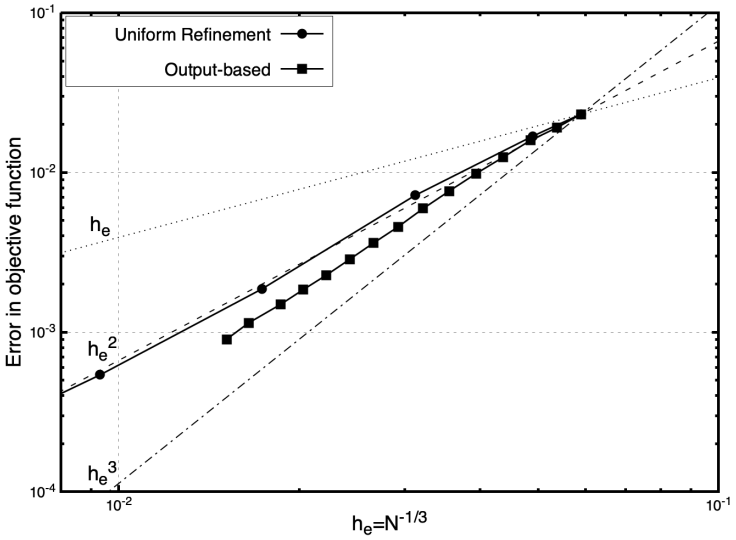
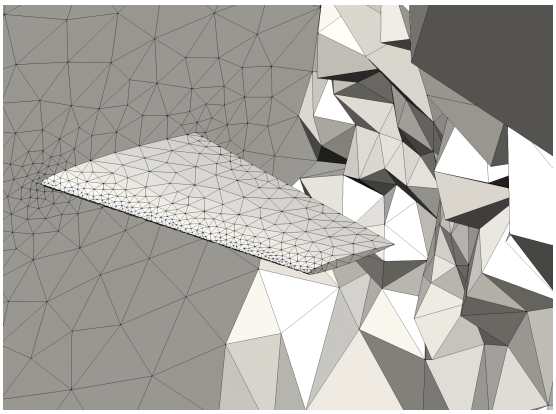
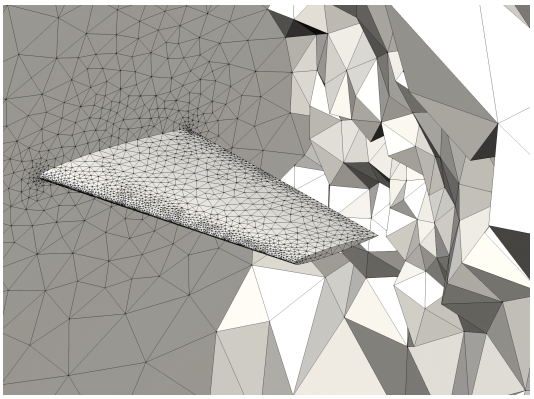


Figure 5.24: Comparison of convergence of error in drag coefficient between uniformly refined, output-based adapted, and output-based corrected meshes for transonic M6 wing

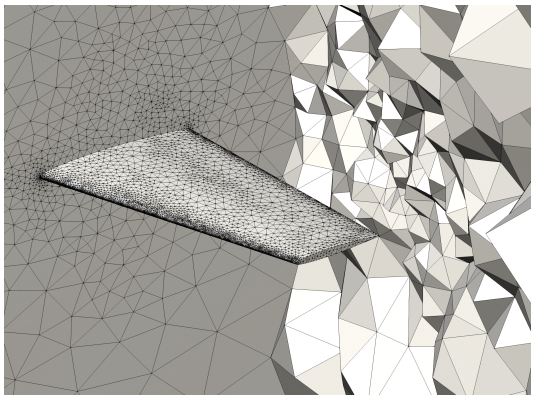
Figures 5.25a-5.25e show 5 refinement steps. Similarly to the subsonic example, first the leading and trailing edges are refined. In adaptation step 7, the flow solver starts to capture expansion shock downstream of the top wing surface; mesh around the wing tip is also being refined. In step 10 and beyond a lambda shock wave is being refined and in step 14 clear refinement regions are present. The wing tip is also strongly targeted, similar to the volume region downstream of the wing tip that allows tip vortex to be captured.



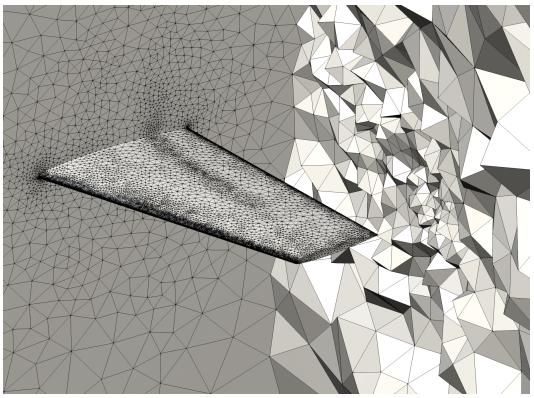
(a) Step 0 (4,882 nodes)



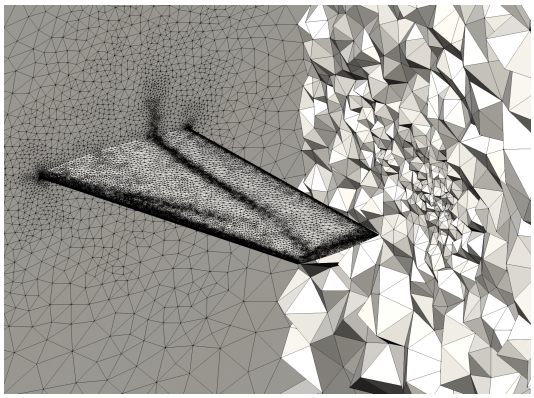
(b) Step 4 (16,331 nodes)



(c) Step 7 (40,143 nodes)



(d) Step 10 (91,900 nodes)

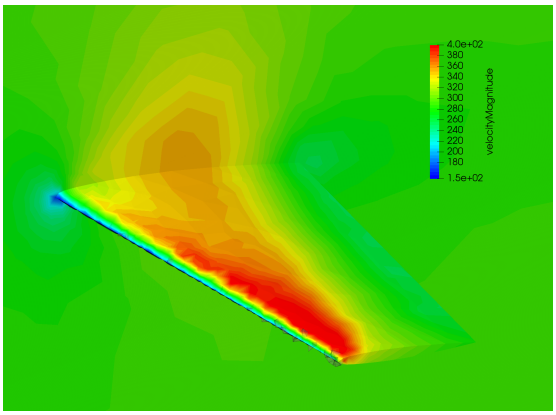


(e) Step 14 (289,156 nodes)

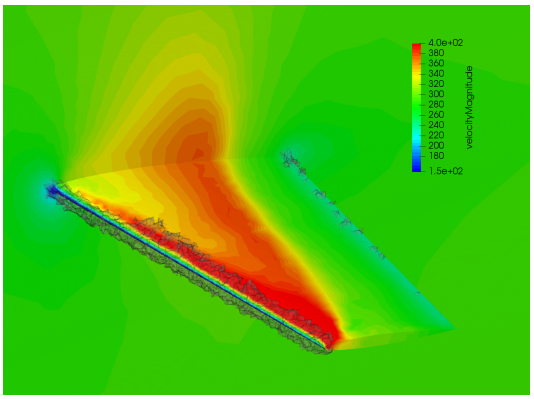
Figure 5.25: Mesh evolution for transonic M6 wing - wing, symmetry plane, and cut-plane downstream the wing

Figure 5.26 shows velocity contours on a set of output-based adapted grids. The initial mesh is not resolving the shock waves, but as the refinement progresses the shock lines are captured more accurately and the velocity magnitude increases as a result of reduced numerical diffusion. A semi-transparent volume structure highlights regions within the computational domain with the smallest cell sizes.

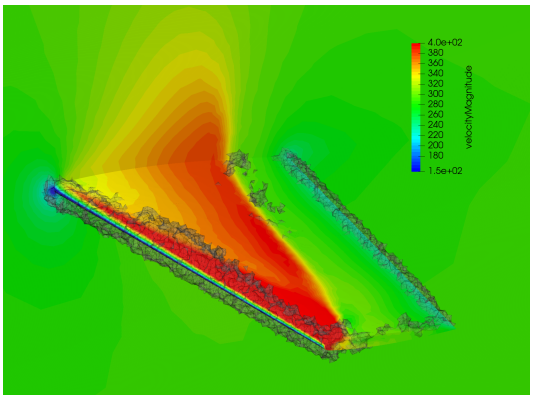
In a similar fashion to the subsonic case, the threshold value is set to 10 mm^3, which corresponds to an edge length of isotropic cell of around 2.2 mm. Unlike the subsonic example the wing top surface is refined; a lambda-shape structure is formed to capture shock waves.



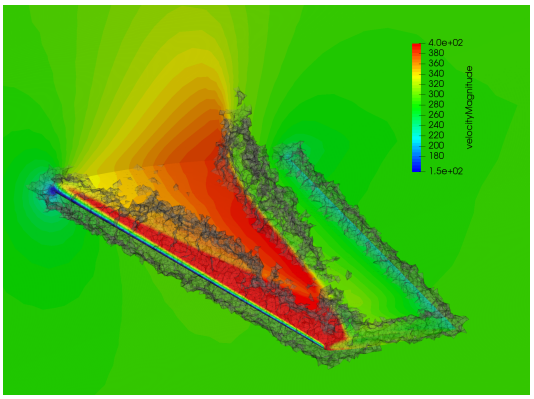
(a) Step 0



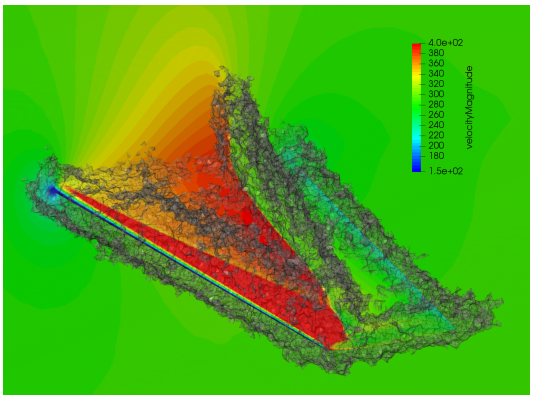
(b) Step 4



(c) Step 7



(d) Step 10

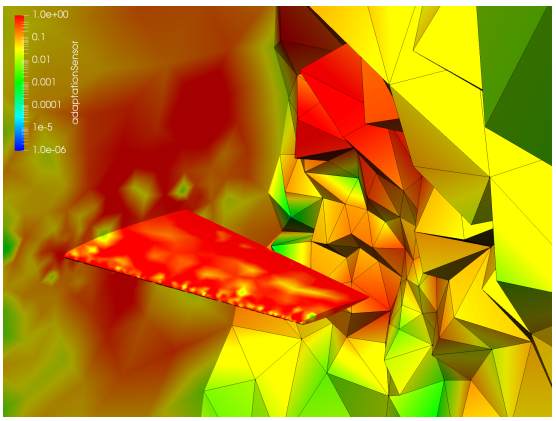


(e) Step 14

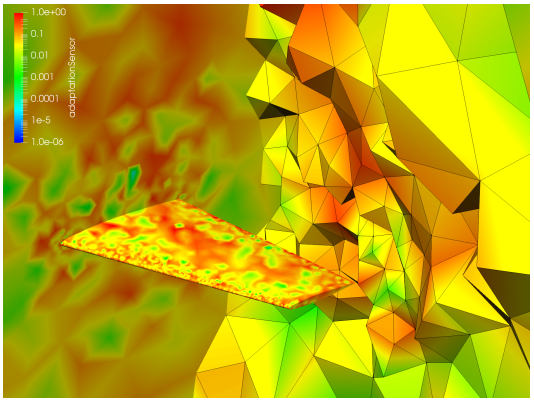
Figure 5.26: Mesh adaptation and evolving solution for transonic M6 wing (velocity contours). Semi-transparent iso-volume marks finest cells (volume below a threshold of 10 mm^3 - cell edge length of around 2.2 mm)

In Figure 5.27a The initial output sensor values on the wing are near the

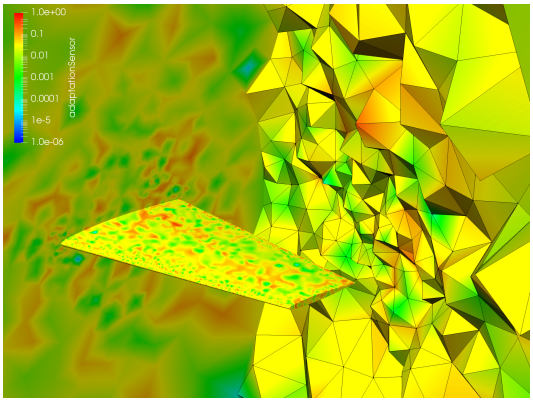
value of 1. At the last adaptation step the level of error reduces below 10^{-3}. A logarithmic scale is used.



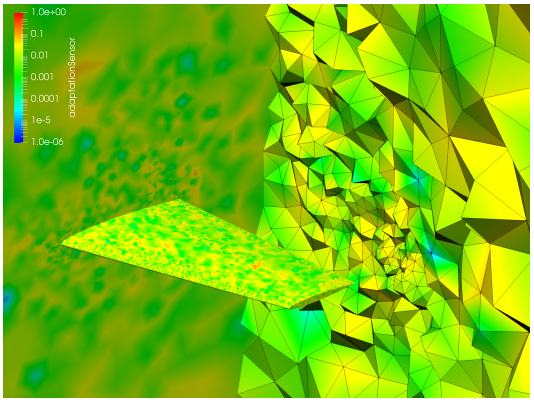
(a) Step 0



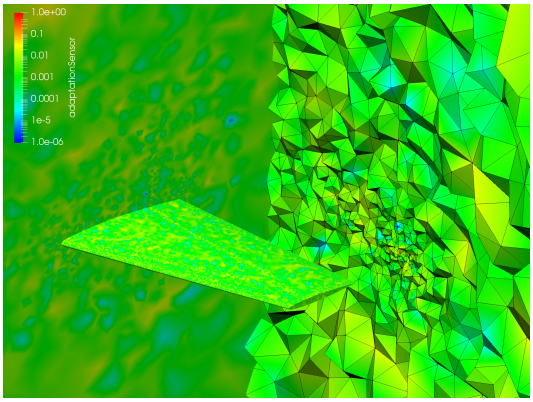
(b) Step 4



(c) Step 7



(d) Step 10



(e) Step 14

Figure 5.27: Adaptation sensor evolution for transonic M6 wing (logarithmic scale) - wing, symmetry plane, and cut-plane downstream the wing

Chapter 6

Conclusion

This thesis has focused on three main aspects. The first area of focus was CFD flow and adjoint solver in STAMPS. A detailed description of the spatial and temporal discretisation of the RANS equations was presented. The efficient cell-based gradient computation method that is consistently accurate for a general mesh type was implemented and verified. Although the cell-based method to calculate gradients is standard, an original approach was proposed for its derivation. With this approach a symbolic mathematical toolbox is used for the derivation and reduction of the number of floating point operations required for the gradient computation. The accurate gradient calculation method allows the achievement of a linearly transparent solution reconstruction and interpolation between multigrid mesh levels. The verification methodology based on a method of manufactured solution was presented and used for the inviscid flow solver testing. A verification results of the STAMPS discretisation scheme showed a reduced order of accuracy for a general mesh type. This inaccuracy originates from the edge-based integration scheme which is a standard for vertex-centred solvers. The edge centroid is used as a flux face integration point, which for a general mesh type is not a Gauss point. Hence, integration is not exact.

Implementation details of the tangent-linear and adjoint solvers were provided, and the usage/benefits of the Tapenade automatic differentiation tool were explained. The routine wrappers created by the author allowed the build process of the STAMPS code to be automated with no additional scripting or code preparations required. This indicates that a modification in the primal code can be

automatically included in the adjoint and tangent-linear solvers using a standard code re-compilation.

The second area of focus was on an automatic CFL-adjustment technique for the STAMPS implicit solver. The results show the benefit of the methodology, which allows almost constant solver run-time to be achieved regardless of initial conditions or CFL number. The automatic CFL-adjustment led to a robust implicit solver with no user interaction related to tedious (manual) CFL number adjustments. The solution recovery step was implemented as a safeguarding technique for solver divergence which can occur where the initial CFL number is too high or ramping is too rapid. Using the RAE282 aerofoil test case, it was shown that the recovery step works effectively.

Additionally, a Jacobian/preconditioner re-computation control (A-CTRL) based on the residual-norm-convergence-history analysis was proposed. The A-CTRL method, combined with automatic CFL-adjustment, led to a 10-20% reduction in the overall run time as compared to the baseline approach. Furthermore, several convergence acceleration techniques taken from the literature were studied. The residual-based time stepping (RBTS), which is a form of local time-stepping, showed that for some cases a more contractive system matrix can be achieved which lead to an improved convergence rate. Similarly for convergence acceleration for highly stretched meshes AR that can enhance convergence for turbulent applications with low values of $y+$; a non-dimensional distance to wall used in turbulence modelling near the wall. However, no major run time reductions were recorded when using either AR or RBTS methods.

Low Mach number scaling was introduced to reduce the stiffness of the RANS system matrix present for compressible solvers. The U-bend example showed an improvement in convergence slope and allowed faster convergence to be achieved by a factor of 2-3.

All implemented methodologies were applied using the three test cases: inviscid M6 wing, turbulent RAE2822 aerofoil, and U-bend channel. Overall, it was shown that the run-time of the flow solver can be reduced by a factor of 0.1 to 0.8. A larger improvements are usually obtained for more complex cases and

flow conditions, e.g. the M6 transonic wing or the U-bend turbulent case. The developments described in Chapter 4 are particularly important for shape optimisation where the run-time reductions are crucial for an efficient process. Solving flow accounts for around half of the overall run-time of the optimisation. Hence, the obtained flow solver run-time reduction by a factor of 0.1 to 0.8 can lead to the overall optimisation time being decreased by a factor of 0.05 to 0.4, which is of particular importance for industrial cases where a reduction of computational cost allows the design time to be decreased.

The final area of focus for this work was an efficient procedure for truncation error estimation that exploits the building blocks of a geometric multi-grid solver. First, the methodology was presented and tested using a method of manufactured solution on a cube domain. Using the proposed method, the truncation error can be estimated with almost no additional cost as compared to other known methods that usually rely on finer grids that are computationally more expensive. Moreover, the methodology can use topologically inconsistent multigrid meshes, which makes it more general. The results in Section 5.4 show that despite the above the estimated truncation error can be effective in driving the mesh adaptation process. The derivation of output error and two adaptation sensors: a) truncation-error-based sensor b) output-error-based sensor was shown. The latter is of primary interest as it indicates regions of the computational domain where not only are the errors large but also where the errors influence an objective function, i.e. an engineering quantity of interest.

Three applications of output-based mesh adaptation were presented in this work. First, a 3D cube case with manufactured solution was used. A Rolls-Royce proprietary code Hydra was exploited for flow and adjoint calculation, as well as for error estimation. At each refinement step, the computational mesh was adapted manually using BoxerMesh and iso-volumes formed by the cells with highest adaptation sensor value. The result showed almost an order of magnitude reduction of the mesh size after just 2 adaptation steps as compared to the uniformly refined grid, while still maintaining the accuracy of the objective function.

Next, a more practical adaptation example was presented using the Onera M6 wing at subsonic flow conditions (Euler flow). The objective function used for output-based mesh adaptation was drag coefficient which for the used flow conditions corresponds to the induced drag due to the non-zero angle of attack. The error in objective function for the output-based adapted mesh converged at an order of 3 whereas the error for the uniformly refined grid converged at an order of 2.

Finally, another adaptation application with the Onera M6 wing was performed. In this example, transonic flow conditions were used. Although the improvement was worse for this case compared to the subsonic example, the errors in objective function showed around 0.4 order increase in convergence rate for the adapted mesh compared to the uniformly refined grid.

All examples of output-based mesh adaptation presented in this work showed improved convergence of error in the objective function of interest. Hence, the proposed methodology has been proved to work in practical applications. Furthermore, the use of two separate CFD codes for mesh adaptation shows that the methodology can be easily implemented in geometric multigrid solvers.

Future work

The results obtained in this work suggest that a modification of the discretisation approach used in STAMPS is required to achieve a scheme that is consistently 2^{nd}-order accurate on a general mesh type, including boundaries. In this work, it was found that the reduced accuracy on a general mesh type originates from the standard edge-based integration scheme. Achieving an exact integral requires a flux face integration point to be a Gauss point, whereas in the current implementation, an edge midpoint was used instead. Further work is required to find an appropriate method to create a 2^{nd}-order accurate solver for a general mesh type while maintaining an efficient edge-based data structure used in STAMPS. A consistently accurate discretisation scheme could lead to a better quality of error estimation and allow discrepancies due to the cell types to be avoided. Furthermore, in the current work only the accuracy of the inviscid solver was investigated

and a verification of the turbulent solver is still required. For this purpose, a manufactured solution for the Spalart-Allmaras turbulence model presented by Roy et al. [2] could be used.

In STAMPS, a comparison of sensitivities obtained using adjoint, tangent, and finite difference approaches was done by Christakopolous [129]. A more rigorous test should be performed to verify the differentiated solver and shape sensitivity. One approach is to use a known solution and perform a convergence study of the error norm on a sequence of uniformly refined grids. In such a study, the verification is successful if the convergence slope of the error norm of adjoint gradients aligns with the convergence slope of the error in the objective function. An example was provided by Nemec and Aftosmis [75]. Another method is to use the Taylor series reminder test as presented by Farrell et al. [76]. For this purpose, a Taylor expansion for an example functional $L(U)$ with a small perturbation δU is introduced, as shown in Eq. 6.1

$$L(U + h_e \delta U) = L(U) + h_e \delta U^T \nabla U + \dots \tag{6.1}$$

Eq. 6.1 can be rearranged to obtain Eq. 6.2, where the gradient ∇U is computed using an adjoint solution.

$$\left| L(U + h_e \delta U) - L(U) - h_e \delta U^T \nabla U \right| \longrightarrow 0 \quad at\ O(|h_e|^2) \tag{6.2}$$

The verification of the gradient computed using an adjoint solution is successful when the term shown in Eq. 6.2 (the Taylor series reminder) converges at second order as $h_e \to 0$. This methodology does not require a sequence of computational meshes nor a known exact solution as compared to the method presented by Nemec and Aftosmis [75]. Hence, it is a recommended approach to be used for a future verification work of the adjoint solver in STAMPS.

Further testing is also recommended for automatic CFL-adjustment and other code enhancements to analyse their benefit for more practical, medium and large scale applications. The methodologies used in this work should be compared to

other state of the art methods to investigate benefits they can offer. Additional research into the optimisation of user-defined parameters for automatic CFL adjustment and constants in the Jacobian re-computation control would also be beneficial.

Mesh adaptation driven by an output-based sensor was applied to 3 inviscid cases in the current work. It would be useful to conduct further testing of the proposed error estimation methodology for turbulent cases, as well as more industrial level applications.

In this work, the first mesh adaptation example was carried out manually using a simple 3D cube case with a Hydra CFD solver and BoxerMesh. In the second and third application the mmg3d tool was used, and the adaptation process was automated. To investigate the optimal setup, different options for an adaptation strategy could be explored, as well as carrying out a study of constants such as the mesh coarsening ratio of hip (default 2.2), or grid size growth rate in mmg3d (default 1.5).

In all presented examples an isotropic mesh adaptation was used. A further improvement in convergence of errors in the objective function may be possible through the use of anisotropic mesh adaptation. This is possible with mmg3d [7, 88].

An output-based adaptation of the transonic Onera M6 wing showed reduced improvements in the rate of converging functional error as compared to the subsonic example (transonic - $O(h_e^{2.4})$, subsonic - $O(h_e^3)$). These results should be investigated further. First, an assessment of order of accuracy of spatial discretisation scheme with slope limiter should be performed. Although the error in drag coefficient was decreasing at second order rate for the uniformly refined mesh (transonic M6 wing), which suggests that the limiter does not largely affect the accuracy of the spatial discretisation scheme, a more rigorous test would be beneficial. Second, the adjoint solver with active slope limiter should be verified as its accuracy may affect the accuracy of an output-based sensor.

After analysing current adaptation methods on various examples (inviscid, viscous) and developing the best refinement strategy, the mesh adaptation process

could be coupled with adjoint-based shape optimisation. The results should be analysed firstly by comparing the overall run-time of the optimisation with and without an output-based mesh adaptation, and then by assessing the influence of the mesh adaptation in optimisation process on the final optimised shape.

Finally, a truncation-error-based adaptation functionality could be implemented in STAMPS so as to allow for automatic mesh refinement as the solution progresses. For geometric multigrid solvers, such an approach is possible as the estimation of truncation error is computationally inexpensive, and could be beneficial in practical engineering applications to reduce computational time and improve solution accuracy.

Appendix A

A.1 STAMPS constants

Constants and reference conditions in STAMPS:

R	Specific gas constant, 287.14285714285711 $\left[\frac{\mathrm{J}}{\mathrm{kg\ K}}\right]$
γ	Heat capacity ratio c_p/c_v, 1.4 [−]
c_p	Specific heat capacity at the constant pressure, $\frac{\gamma \mathrm{R}}{\gamma-1}$ $\left[\frac{\mathrm{J}}{\mathrm{kg\ K}}\right]$
c_v	Specific heat capacity at the constant volume, $\frac{\mathrm{R}}{\gamma-1}$ $\left[\frac{\mathrm{J}}{\mathrm{kg\ K}}\right]$
T_{ref}	Reference temperature, 300 [K]
$\mu_{L,ref}$	Ref. laminar dynamic viscosity at T_{ref}, $1.633920236187 \cdot 10^{-5}$ $\left[\frac{\mathrm{kg}}{\mathrm{m\ s}}\right]$
C_S	Southerland's constant, 110.0 [K]
Pr	Prandtl number, 0.713 [−]
Pr_t	Turbulent Prandtl number, 0.85 [−]

Spalart-Allmaras turbulence model constants:

C_{b1}	0.1355
C_{b1}	0.1355
C_{b2}	0.622
κ	0.41
σ	2/3
C_{w2}	0.3
C_{w3}	2
C_{v1}	7.1

C_{t3}	1.2
C_{t4}	0.5
C_2	0.7
C_3	0.9

A.2 RANS equations in STAMPS

The integral form of the compressible RANS system of equations with Spalart-Allmaras turbulence model [144] is presented in Eq. A.2. This system is derived using the differential form of Navier-Stokes system of equations and the divergence theorem - A.1, where $\vec{F}$ is a vector field and $\vec{n} = [n_x, n_y, n_z]^T$ is normal defining face of the closed control volume. Refer to e.g. [118, 102] for more details on derivation.

$$\iiint_{\Omega} \nabla \cdot \vec{F} d\Omega = \oiint_{S} \vec{F} \cdot \vec{n} dS \tag{A.1}$$

All flow variables present in the equation are Reynolds-averaged quantities - see section 7.1.1 in [102] for details on Reynolds-averaging. All volume sources are kept in the system of equations shown in Eq. A.2 for generality, and all except for SA_{src} are assumed zero. All model and material constants are defined in Section A.1.

Continuity equation : (A.2)

$$\frac{\partial}{\partial t}\int_{\Omega}\rho\, d\Omega + \oint_{\partial\Omega}\rho V\, dS = \int_{\Omega} f_{\rho}\, d\Omega$$

Momentum equations :

$$\mathbf{x}: \quad \frac{\partial}{\partial t}\int_{\Omega}\rho u\, d\Omega + \oint_{\partial\Omega}\left(\rho u V + n_x p\right)\, dS -$$
$$-\oint_{\partial\Omega}\left(n_x\tau_{xx}^{eff} + n_y\tau_{yx}^{eff} + n_z\tau_{zx}^{eff}\right) dS = \int_{\Omega} f_{\rho u}\, d\Omega$$

$$\mathbf{y}: \quad \frac{\partial}{\partial t}\int_{\Omega}\rho v\, d\Omega + \oint_{\partial\Omega}\left(\rho v V + n_y p\right)\, dS -$$
$$-\oint_{\partial\Omega}\left(n_x\tau_{xy}^{eff} + n_y\tau_{yy}^{eff} + n_z\tau_{zy}^{eff}\right) dS = \int_{\Omega} f_{\rho v}\, d\Omega$$

$$\mathbf{z}: \quad \frac{\partial}{\partial t}\int_{\Omega}\rho w\, d\Omega + \oint_{\partial\Omega}\left(\rho w V + n_z p\right)\, dS -$$
$$-\oint_{\partial\Omega}\left(n_x\tau_{xz}^{eff} + n_y\tau_{yz}^{eff} + n_z\tau_{zz}^{eff}\right) dS = \int_{\Omega} f_{\rho w}\, d\Omega$$

Energy equation :

$$\frac{\partial}{\partial t}\int_{\Omega}\rho e\, d\Omega + \int_{\partial\Omega}\rho h_t V\, dS -$$
$$-\oint_{\partial\Omega}\left(n_x\Theta_x^{eff} + n_y\Theta_y^{eff} + n_z\Theta_z^{eff}\right) dS = \int_{\Omega} f_{\rho e}\, d\Omega$$

Spalart-Allmaras equation :

$$\frac{\partial}{\partial t}\int_{\Omega}\widehat{\nu} d\Omega + \oint_{\partial\Omega}\widehat{\nu} V\, dS - \oint_{\partial\Omega}\frac{1}{\sigma}\left(\nu_L + \widehat{\nu}\right)\left(\nabla\widehat{\nu}\cdot\vec{n}\right)\, dS = \int_{\Omega} S A_{src}\, d\Omega$$

The ideal gas law (Eq. A.3) is used to close the system:

$$p = \rho R T \tag{A.3}$$

V is the contravariant velocity and $\vec{U}_v$ is the vector of x, y, z-velocity components (Eq. A.4):

$$V = \vec{U}_v\cdot\vec{n} = \left(n_x u + n_y v + n_z w\right), \quad \vec{U}_v = [u, v, w]^T = [u_x, u_y, u_z]^T \tag{A.4}$$

The effective stress tensor components derived for the RANS system of equations with a closure turbulent viscosity term μ_t are defined using Eq. A.5. Note that

$[u_x, u_y, u_z] = [u, v, w]$ is an alternative notation used for velocity components.

$$\tau_{i,j}^{eff} = (\mu + \mu_t)\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right) + (\lambda + \lambda_t)\, div(U)\, \delta_{i,j}, \qquad (A.5)$$
$$i = j \;\rightarrow\; \delta_{i,j} = 1; \quad i \neq j \;\rightarrow\; \delta_{i,j} = 0$$

The turbulent viscosity is estimated using SA variable $\widehat{\nu}$ resulting from the SA equation, and λ *and* λ_t is obtained based on the Stokes assumption:

$$\mu_t = \rho\widehat{\nu} f_{v1}, \quad \nu_t = \widehat{\nu} f_{v1}, \quad \lambda = -\frac{2}{3}\mu_L, \quad \lambda_t = -\frac{2}{3}\mu_t \qquad (A.6)$$

Parameter f_{v1} is defined as follows:

$$f_{v1} = \frac{\chi^3}{\chi^3 + C_{v1}^3}, \quad \chi = \frac{\widehat{\nu}}{\nu_L} \qquad (A.7)$$

The convective term in energy Eq. A.2 is written using specific total enthalpy h_t for a more compact form. The total enthalpy per unit mass is defined as presented in Eq. A.8.

$$h_s = \frac{\gamma}{\gamma - 1}\,\frac{p}{\rho} + \frac{u^2 + v^2 + w^2}{2} \qquad (A.8)$$

This equation can be derived using Eq. A.9, where the total enthalpy per unit mass is expressed in various forms with i_e being the specific internal energy, e_k specific kinetic energy, and h_s specific enthalpy. Replacing i_e and e_k with relations from Eq. A.10 and transforming obtained formula using dependencies from Eq. A.11 leads to the final form presented in Eq. A.8.

$$h_t = e + \frac{p}{\rho} = h_s + e_k = i_e + e_k + \frac{p}{\rho} \qquad (A.9)$$

$$i_e = c_v T = c_v \frac{p}{\rho R} \qquad (A.10)$$
$$e_k = 0.5\left(u^2 + v^2 + w^2\right)$$
$$e = i_e + e_k, \quad h_s = i_e + \frac{p}{\rho}$$

Eq. A.11 shows the relation between the specific heat capacity at constant pressure/volume (c_p/c_v), and the ideal gas constant R. Variable γ is a heat capacity ratio.

$$\frac{c_p}{c_v} = \gamma, \quad c_p - c_v = R \tag{A.11}$$

The Θ_i^{eff} terms present in the energy equation are as follows:

$$\begin{aligned}
\Theta_x^{eff} &= u\tau_{xx}^{eff} + v\tau_{xy}^{eff} + w\tau_{xz}^{eff} + (k_L + k_t)\frac{\partial T}{\partial x} \\
\Theta_y^{eff} &= u\tau_{yx}^{eff} + v\tau_{yy}^{eff} + w\tau_{yz}^{eff} + (k_L + k_t)\frac{\partial T}{\partial y} \\
\Theta_z^{eff} &= u\tau_{zx}^{eff} + v\tau_{zy}^{eff} + w\tau_{zz}^{eff} + (k_L + k_t)\frac{\partial T}{\partial z}
\end{aligned} \tag{A.12}$$

Laminar thermal conductivity k_L and turbulent thermal conductivity k_t are defined in Eq. A.13. See Section A.1 for definitions of constants.

$$k_L = \mu_L c_p / Pr, \quad k_t = \mu_t c_p / Pr_t \tag{A.13}$$

The Spalart-Allmaras source term is defined in Eq. A.14. All model constants are defined in Section A.1.

$$SA_{src} = C_{b1}(1 - f_{t2})\widehat{S}\,\widehat{\nu} + \frac{C_{b2}}{\sigma}\left(\frac{\partial \widehat{\nu}}{\partial x_j}\right)^2 - \left[C_{w1} f_w - \frac{C_{b1}}{\kappa^2} f_{t2}\right]\left(\frac{\widehat{\nu}}{d}\right)^2 \tag{A.14}$$

The remaining variables are defined in Eqs. A.15-A.17. Note that variable χ was already defined in Eq. A.7.

$$C_{w1} = \frac{C_{b1}}{\kappa^2} + \frac{1 + C_{b2}}{\sigma} \tag{A.15}$$

$$f_{v2} = 1 - \frac{\chi}{1 + \chi f_{v1}}, \quad f_{t2} = C_{t3}\, exp\left(-C_{t4}\,\chi^2\right) \tag{A.16}$$

$$f_w = g\left[\frac{1 + C_{w3}^6}{g^6 + C_{w3}^6}\right]^{1/6}, \quad g = r + C_{w2}\left(r^6 - r\right), \quad r = \frac{\chi^3}{\chi^3 + C_{v1}^3} \tag{A.17}$$

According to Allmaras and Johnson [117], the vorticity $\widehat{S}$ is modified to prevent negative values which are nonphysical and can lead to convergence problems. The modification is presented in Eq. A.18 with $\overline{S}$ defined in Eq. A.19 and the original

vorticity S in Eq. A.20. Variable d stands for the distance to wall.

$$\widehat{S} = \begin{cases} S + \overline{S} & : \quad \overline{S} \geq -c_{v2} S \\ S + \frac{S\left(c_{v2}^2 S + c_{v3} \overline{S}\right)}{(c_{v3} - 2c_{v2})S - \overline{S}} & : \quad \overline{S} < -c_{v2} S \end{cases} \tag{A.18}$$

$$\overline{S} = \frac{\widehat{\nu}}{\kappa^2 d^2} f_{v2} \tag{A.19}$$

$$S = W + f_{v2} \frac{\widehat{\nu}}{\kappa^2 d^2}, \quad W = \sqrt{2 W_{i,j} W_{i,j}}, \quad W_{i,j} = 0.5 \left(\frac{\partial u_i}{\partial x_j} - \frac{\partial u_j}{\partial x_i} \right) \tag{A.20}$$

Vorticity magnitude W is defined in Eq. A.21 using Einstein index notation, so that it expands to:

$$W = \sqrt{2 \left(W_{xx}^2 + W_{xy}^2 + W_{xz}^2 + W_{yx}^2 + W_{yy}^2 + W_{yz}^2 + W_{zx}^2 + W_{zy}^2 + W_{zz}^2 \right)} \tag{A.21}$$

Laminar viscosity is defined as a function of temperature using Southerland's law - Eq. A.22 with reference quantities defined in Section A.1.

$$\mu_L = \mu_{L,ref} \left(\frac{T}{T_{ref}} \right)^{3/2} \frac{T_{ref} + C_S}{T + C_S} \tag{A.22}$$

A.3 ROE averaged variables

The ROE averages at the flux face are calculated using left L and right R states as shown in Eq. A.23.

$$
\begin{aligned}
\widetilde{\rho} &= \sqrt{\rho_L \rho_R} \\
\widetilde{u} &= \frac{u_L\sqrt{\rho_L} + u_R\sqrt{\rho_R}}{\sqrt{\rho_L} + \sqrt{\rho_R}} \\
\widetilde{v} &= \frac{v_L\sqrt{\rho_L} + v_R\sqrt{\rho_R}}{\sqrt{\rho_L} + \sqrt{\rho_R}} \\
\widetilde{w} &= \frac{w_L\sqrt{\rho_L} + w_R\sqrt{\rho_R}}{\sqrt{\rho_L} + \sqrt{\rho_R}} \\
\widetilde{h_t} &= \frac{h_{t,L}\sqrt{\rho_L} + h_{t,R}\sqrt{\rho_R}}{\sqrt{\rho_L} + \sqrt{\rho_R}} \\
\widetilde{c} &= \sqrt{(\gamma - 1)\left(\widetilde{h_t} - \widetilde{q}^2/2\right)} \\
\widetilde{V} &= \widetilde{u}n_x + \widetilde{v}n_y + \widetilde{w}n_z \\
\widetilde{q}^2 &= \widetilde{u}^2 + \widetilde{v}^2 + \widetilde{w}^2
\end{aligned}
\tag{A.23}
$$

A.4 Coefficient computation for cell-based gradient

In order to efficiently compute coefficient matrix C for each basic element type presented in Figure 2.1, the symbolic maths toolbox of Matlab[28] or Python SymPy[29] packages can be used.

[28] http://uk.mathworks.com/products/symbolic/?requestedDomain=www.mathworks.co
[29] http://www.sympy.org/en/index.html

Algorithm 7 Symbolic Green-Gauss integration

1: **do** $i_f = 1,\ N_f(i_c)$
2: **if** triangle **then**
3: $\vec{X} = \vec{X}_i + s\left(\vec{X}_j - \vec{X}_i\right) + t\left(\vec{X}_k - \vec{X}_i\right) + st\left(\vec{X}_i - 0.5\vec{X}_j - 0.5\vec{X}_k\right)$
4: $F = F_i + s\left(F_j - F_i\right) + t\left(F_k - F_i\right) + st\left(F_i - 0.5F_j - 0.5F_k\right)$
5: **else if** quadraliteral **then**
6: $\vec{X} = \vec{X}_i + s\left(\vec{X}_j - \vec{X}_i\right) + t\left(\vec{X}_l - \vec{X}_i\right) + st\left(\vec{X}_i + \vec{X}_k - \vec{X}_j - \vec{X}_l\right)$
7: $F = F_i + s\left(F_j - F_i\right) + t\left(F_l - F_i\right) + st\left(F_i + F_k - F_j - F_l\right)$
8: **end if**
9: $\frac{\partial \vec{X}}{\partial s} = jacobian\left(\vec{X}, s\right)$
10: $\frac{\partial \vec{X}}{\partial t} = jacobian\left(\vec{X}, t\right)$
11: $\vec{faceContribution} = int\left[int\left[\left(F\ \frac{\partial \vec{X}}{\partial s} \times \frac{\partial \vec{X}}{\partial t}\right), s, 1, 0\right], t, 1, 0\right]$
12: $\vec{Integral}_{GG} = \vec{Integral}_{GG} + \vec{faceContribution}$
13: **End do**

The basic idea is to use shape functions to describe linearly varying field on triangle face (Algorithm 7 line 3 and 4) and quadrilateral face (Algorithm 7 line 6 and 7), perform symbolic integration using Green-Gauss approach and extract coefficient vectors $\vec{C}_i$ at each node. The detailed procedure is presented in Algorithm 7. The matlab function '$jacobian(...)$' is used to symbolically calculate required derivatives, and function '$int(...)$' to perform symbolic integration. $\vec{X}$ is a vector of coordinates (x, y, z), the variable $N_f(i_c)$ is a list of faces forming given element and indices (i, j, k, l) to the nodes forming the face. The coefficient matrix C can be extracted from the obtained integral by rearranging it with respect to the field F - Eq. A.24. The collect function of Matlab can be used for this purpose. The variable N_n is a list of nodes forming an element i.e. Tet: 4, Pyramid: 5, Prism: 6, Hex: 8.

$$\vec{Integral}_{GG} = \begin{bmatrix} c_{x,1}F_1 + c_{x,2}F_2 + ... + c_{x,N_n}F_{N_n} \\ c_{y,1}F_1 + c_{y,2}F_2 + ... + c_{y,N_n}F_{N_n} \\ c_{z,1}F_1 + c_{z,2}F_2 + ... + c_{z,N_n}F_{N_n} \end{bmatrix} \tag{A.24}$$

Note that the symbolic integration can also be performed using pure geometrical approach, which produces a final coefficient matrix that is identical to the one

obtained with the method described in Algorithm 7 - verified by the author using symbolic subtraction of integral formulas for both approaches that reduced to $[0,0,0]^T$. To achieve this, any quadrilateral face of each element type has to be triangulated in the way presented on Figure A.1, where $\vec{X}_{i_c} = 0.25(\vec{X}_i + \vec{X}_j + \vec{X}_k + \vec{X}_l)$ is quadrilateral centroid. The integral can then be calculated accurately for linear fields using Eq. A.25. After symbolic integration and rearrangements as presented in Eq. A.24 the coefficients can be extracted.

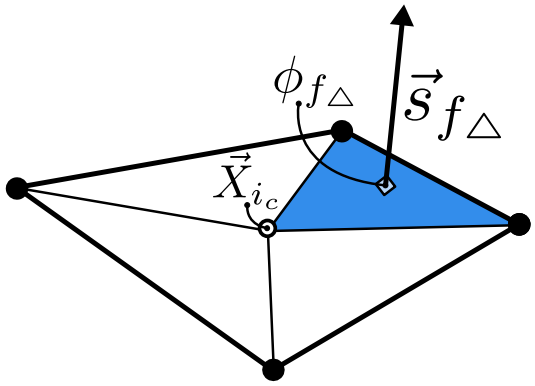


Figure A.1: Accurate integration using geometric approach

$$\vec{Integral}_{GG,i_c} = \sum_{f_\triangle=1}^{N_{f_\triangle}(i_c)} \phi_{f_\triangle} \vec{s}_{f_\triangle} \tag{A.25}$$

While performing symbolic computations, it is important to keep the element connectivity information consistent with those from the solver - see Figure 2.1. The coefficients can be used to calculate gradients that are exact for any linear field ϕ as presented in Section 2.3.5.

A.5 Method of manufactured solution for Euler equations

The 3D compressible Euler equations with a source terms on the right-hand side are used - Eq. A.26.

$$\frac{\partial(\rho u)}{\partial x} + \frac{\partial(\rho v)}{\partial y} = f_\rho(x,y,z) \tag{A.26}$$

$$\frac{\partial(\rho u^2 + p)}{\partial x} + \frac{\partial(\rho uv)}{\partial y} + \frac{\partial(\rho uw)}{\partial z} = f_u(x,y,z)$$

$$\frac{\partial(\rho vu)}{\partial x} + \frac{\partial(\rho v^2 + p)}{\partial y} + \frac{\partial(\rho vw)}{\partial z} = f_v(x,y,z)$$

$$\frac{\partial(\rho wu)}{\partial x} + \frac{\partial(\rho wv + p)}{\partial y} + \frac{\partial(\rho w^2 + p)}{\partial z} = f_v(x,y,z)$$

$$\frac{\partial(\rho u e_t + pu)}{\partial x} + \frac{\partial(\rho v e_t + pv)}{\partial y} + \frac{\partial(\rho w e_t + pw)}{\partial z} = f_e(x,y,z)$$

The specific total energy e_t is given by the Eq. A.27 and the Euler equations are closed with the ideal gas law (Eq. A.28).

$$e_t = \frac{1}{\gamma - 1} RT + \frac{u^2 + v^2 + w^2}{2} \tag{A.27}$$

$$p = \rho RT \tag{A.28}$$

The supersonic manufactured solution [41] extended to 3D is presented in Eq. A.29, where the constants are defined in table (A.1). The same set of constants was used in [1].

$$\rho(x,y,z) = \rho_0 + \rho_x \sin\left(\frac{\alpha_{\rho x}\pi x}{L}\right) + \rho_y \cos\left(\frac{\alpha_{\rho y}\pi y}{L}\right) + \rho_z \sin\left(\frac{\alpha_{\rho z}\pi z}{L}\right) \tag{A.29}$$

$$u(x,y,z) = u_0 + u_x \sin\left(\frac{\alpha_{ux}\pi x}{L}\right) + u_y \cos\left(\frac{\alpha_{uy}\pi y}{L}\right) + u_z \sin\left(\frac{\alpha_{uz}\pi z}{L}\right)$$

$$v(x,y,z) = v_0 + v_x \cos\left(\frac{\alpha_{vx}\pi x}{L}\right) + v_y \sin\left(\frac{\alpha_{vy}\pi y}{L}\right) + v_z \sin\left(\frac{\alpha_{vz}\pi z}{L}\right)$$

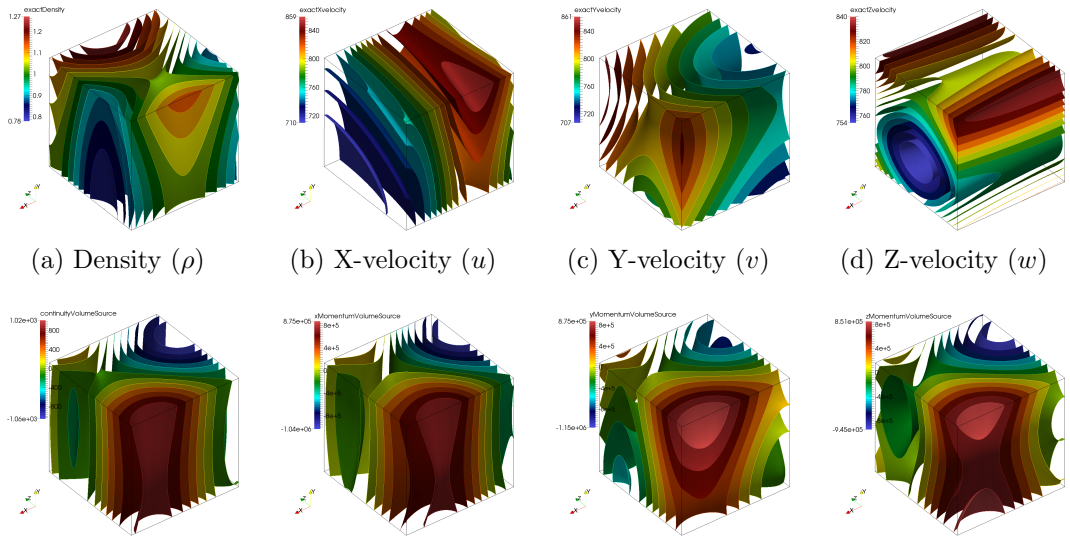
$$w(x,y,z) = w_0 + w_x \cos\left(\frac{\alpha_{wx}\pi x}{L}\right) + w_y \sin\left(\frac{\alpha_{wy}\pi y}{L}\right) + w_z \sin\left(\frac{\alpha_{wz}\pi z}{L}\right)$$

$$p(x,y,z) = p_0 + p_x \cos\left(\frac{\alpha_{px}\pi x}{L}\right) + p_y \sin\left(\frac{\alpha_{py}\pi y}{L}\right) + p_z \sin\left(\frac{\alpha_{pz}\pi z}{L}\right)$$

Variable, ϕ	ϕ_0	ϕ_x	ϕ_y	ϕ_z	$\alpha_{\phi x}$	$\alpha_{\phi y}$	$\alpha_{\phi z}$
ρ [kg/m^3]	1	0.15	-0.1	-0.12	1	0.5	1.5
u [m/s]	800	50	-30	-18	1.5	0.6	0.5
v [m/s]	800	-75	40	-30	0.5	2/3	1.25
w [m/s]	800	-75	40	35	0.5	2/3	1
p [N/m^2]	1×10^5	0.2×10^5	0.5×10^5	-0.35×10^5	2	1	1/3

Table A.1: Constants for the supersonic manufactured solution [1]

Figure A.2 and 5.8 presents manufactured solution in the square domain $1 \times 1 \times 1$ meter and the corresponding source terms arising from the 'made up' solution. The equations for the sources were derived using the manufactured solution (Eq. A.29) inserted into the Euler system of equations (Eq. A.26). The Python symbolic toolbox - SymPy[30] was used for derivations. The sources could also be derived by hand, however this method is more error prone and tedious.



(a) Density (ρ) (b) X-velocity (u) (c) Y-velocity (v) (d) Z-velocity (w)

(e) Continuity source (f_ρ) (f) X-momentum source (f_u) (g) Y-momentum source (f_v) (h) Z-momentum source (f_w)

Figure A.2: The 3D supersonic manufactured solution and corresponding source terms

[30] http://www.sympy.org/en/index.html

A.6 An example usage of a Tapenade differentiated code

To understand better the usage of differentiated subroutines for matrices the example of a partial derivative of residual R with respect to node coordinates X is presented. The partial derivative $\frac{\partial R}{\partial X}$ is a matrix of the total size $6N \times 3N$ (RANS system of equations with SA turbulence model), where each of the entries in the matrix A.30 is 6×3 block matrix for 6-equations and 3-dimensions. The differentiated subroutine RL defined in Section 3.3 is used for this purpose.

$$\frac{\partial R}{\partial X} = \begin{bmatrix} \frac{\partial R_1}{\partial X_1} & \frac{\partial R_1}{\partial X_2} & \frac{\partial R_1}{\partial X_3} & . & . & . & . & \frac{\partial R_1}{\partial X_n} \\ \frac{\partial R_2}{\partial X_1} & . & & & & & & \\ \frac{\partial R_3}{\partial X_1} & & . & & & & & \\ . & & & . & & & & \\ . & & & & . & & & \\ . & & & & & . & & \\ . & & & & & & . & \\ \frac{\partial R_n}{\partial X_1} & & & & & & & \frac{\partial R_n}{\partial X_n} \end{bmatrix}, \frac{\partial R_i}{\partial X_j} = \begin{bmatrix} \frac{\partial R_{\rho,i}}{\partial x_j} & \frac{\partial R_{\rho,i}}{\partial y_j} & \frac{\partial R_{\rho,i}}{\partial z_j} \\ \frac{\partial R_{\rho u,i}}{\partial x_j} & \frac{\partial R_{\rho u,i}}{\partial y_j} & \frac{\partial R_{\rho u,i}}{\partial z_j} \\ \frac{\partial R_{\rho v,i}}{\partial x_j} & \frac{\partial R_{\rho v,i}}{\partial y_j} & \frac{\partial R_{\rho v,i}}{\partial z_j} \\ \frac{\partial R_{\rho w,i}}{\partial x_j} & \frac{\partial R_{\rho w,i}}{\partial y_j} & \frac{\partial R_{\rho w,i}}{\partial z_j} \\ \frac{\partial R_{\rho e,i}}{\partial x_j} & \frac{\partial R_{\rho e,i}}{\partial y_j} & \frac{\partial R_{\rho e,i}}{\partial z_j} \\ \frac{\partial R_{SA,i}}{\partial x_j} & \frac{\partial R_{SA,i}}{\partial y_j} & \frac{\partial R_{SA,i}}{\partial z_j} \end{bmatrix} \tag{A.30}$$

If the column j of the matrix $\frac{\partial R}{\partial X}$ is to be obtained efficiently in terms of runtime, the forward differentiated subroutine RL_d should be used as shown in the following pseudo-code:

```
1  Xd = 0.0
2  do iDir = 1, 3
3    Xd(1,j) = 1.0
4    subroutine RL_d(X↓, Xd↓, ..., R↑, Rd↑, L↑, Ld↑)
5    dRdX_j(iDir,:,:) = Rd
6    Xd(1,j) = 0.0
7  end do
```

Note that the example shows the derivative obtained for all six equations and for all 3-coordinates $X_j = [x_j, y_j, z_j]^T$ of node j, this is, $\frac{\partial R}{\partial X_j}$.

On the other hand, if the row i of the matrix $\frac{\partial R}{\partial X}$ is to be obtained efficiently, the reverse differentiated subroutine RL_b should be used:

```
1  Lb = 0.0
2  Rb = 0.0
3  do iEqn = 1, 6
4    Rb(iEqn,i) = 1.0
5    subroutine RL_b(X↓, Xb↑, ..., R↑, Rb↓, L↑, Lb↓)
6    dRdX_i(:,iEqn,:) = Xb
7    Rb(iEqn,i) = 0.0
8  end do
```

In this manner the row $\frac{\partial R_i}{\partial X}$ is obtained.

A.7 Weighted explicit Laplacian smoothing

Laplace's equation has the form presented in Eq. A.31. It can be easily discretised using e.g. an explicit time stepping and an edge-based data structure as presented in Eq. A.32. The constant $\beta = 1.0$ is used, which is related to the maximum allowed time step constrained by the stability condition of an explicit scheme. The edge-length-weighting is applied to prevent distortions due to the cell sizes [145].

$$\frac{\partial \phi}{\partial t} = \Delta \phi \tag{A.31}$$

$$\phi_i^{n+1} = \phi_i^n + \beta \left(\sum_{j=1}^{m} \frac{\phi_j^n - \phi_i^n}{l_{ij}} \right) \Bigg/ \sum_{j=1}^{m} l_{ij} \tag{A.32}$$

Figure A.3 shows the eigenvalues of explicit smoothing system matrix for a simple 1D case. Each eigenvalue corresponds to the eigen-vector i.e. shape mode. The magnitude of the eigenvalue tells how much given shape mode will be damped when the smoothing is performed. The graph confirms that the explicit smoothing is very effective in filtering the high-frequency modes while having a minor influence on the the low-frequency modes.

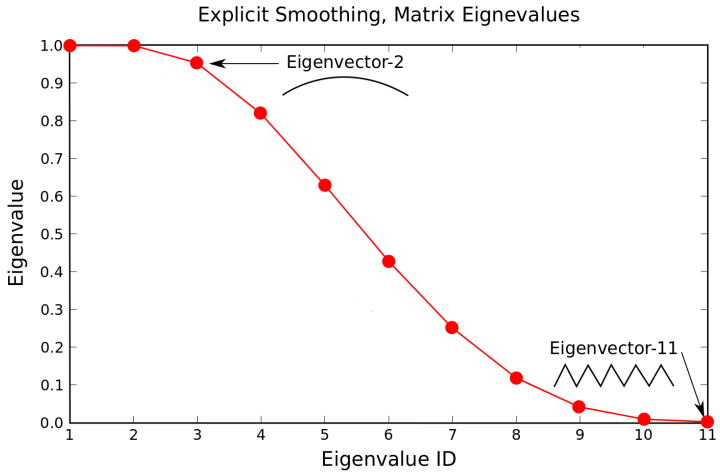


Figure A.3: Eigenvalues of explicit smoothing system matrix for simple 1D case

Bibliography

[1] C. J. Roy, T. M. Smith. and C. C. Ober. Verification of a compressible CFD code using the method of manufactured solutions. In *Proceedings of the 32nd AIAA Fluid Dynamics Conference and Exhibit*, St. Louis, Missouri, 2002.

[2] C. J. Roy, E. Tendean, S. Veluri, R. Rifki, E. Luke, and S. Hebert. Verification of RANS turbulence models in Loci-CHEM using the method of manufactured solutions. *18th AIAA Computational Fluid Dynamics Conference*, Miami, Florida, 2002. doi: 10.2514/6.2007-4203.

[3] R. Courant, K. Friedrichs, and H. Lewy. Über die partiellen Differenzengleichungen der mathematischen Physik. *Mathematische Annalen*, 100(1):32–74, 1928. doi: 10.1007/BF01448839

[4] A. Haselbacher, and J. Blazek. Accurate and efficient discretization of Navier-Stokes equations on mixed grid. *AIAA Journal*, 38(11):2094–2102, 2000. doi: 10.2514/2.871.

[5] R. D. Richtmyer. *Difference methods for initial-value problems.* Krieger Publishing Company, 2nd Revised edition,1994. ISBN: 0894647636.

[6] G. Strang, G. J. Fix. *An Analysis of the Finite Element Method.* WellesleyCambridge Press, Wellesley, Massachusetts, 1988.

[7] E. Gauci, A. Belme, A. Carabias, A. Loseille, F. Alauzet, A. Dervieux. A priori error-based mesh adaptation in CFD. *HAL-Inria*, 2018. url: https://hal.inria.fr/hal-01928249.

[8] L. F. Richardson. *Weather prediction by numerical process.* Cambridge University Press, 2007.

[9] L. F. Richardson. The approximate arithmetical solution by finite differences of physical problems involving differential equations, with an application to the stresses in a masonry dam. *Philosophical Transactions of the Royal Society of London, JSTOR*, 210:307:357, 1911. url: www.jstor.org/stable/90994.

[10] A. Jameson. *Computational Fluid Dynamics: Past, present and future.* Deparment of Aeronautics and Astronautics, Stanford University, 11, 2012.

[11] J. Eckert, J. Mauchly, H. Goldstine, and J. Brainerd. *Description of the ENIAC and comments on electronic digital machines.* Moore School of Electrical Engineering, Philadelphia, Pennsylvania, 1945.

[12] G. Ifrah, E. F. Harding, D. Bellos, S. Wood. *The universal history of computing: From the abacus to quantum computing.* John Wiley and Sons, Inc., 2000.

[13] D. E. Golberg. *Genetic algorithms in search, optimization, and machine learning.* Addison-Wesley Longman Publishing Co., Inc. Boston, MA, 1989. ISBN:0201157675.

[14] L. Davis. *Handbook of genetic algorithms.* Van Nostrand Reinhold, 1st edition, 1991. ISBN: 0442001738.

[15] B. Mohammadi, and O. Pironneau. Shape optimization in fluid mechanics. *Annual Review of Fluid Mechanics*, 36(1):255–279, 2004. doi: 10.1146/annurev.fluid.36.050802.121926.

[16] A. Jameson. Aerodynamic design via control theory. *Journal of Scientific Computing*, 3(3):233–260, 1988. doi: 10.1007/BF01061285.

[17] O. Pironneau. On optimum design in fluid mechanics. *Journal of Fluid Mechanics*, 64(1):97–110, 1974. doi: 10.1017/S0022112074002023.

[18] S. Xu, D. Radford, M. Meyer, and J.-D. Müller. Stabilisation of discrete steady adjoint solvers. *Journal of Computational Physics*, 299:175–195, 2015. doi: 10.1016/j.jcp.2015.06.036.

[19] D. E. Keyes, D. R. Reynolds, and C. S. Woodward. Implicit solvers for large-scale nonlinear problems. *Journal of Physics: Conference Series*, 46:433–442, 2006. doi: 10.1088/1742-6596/46/1/060.

[20] C. Michalak, and C. Ollivier-Gooch. Globalized matrix-explicit Newton-GMRES for the high-order accurate solution of the Euler equations. *Computers and Fluids*, 39(7):1156–1167, 2010. doi: 10.1016/j.compfluid.2010.02.008.

[21] P. F. Fischer. Projection techniques for iterative solution of Ax = b with successive right-hand sides. *Computer Methods in Applied Mechanics and Engineering*, 163(1):193–204, 1998. doi: 10.1016/S0045-7825(98)00012-7.

[22] R. P. Pawlowski, J. N. Shadid, J. P. Simonis, and H. F. Walker. Globalization techniques for Newton-Krylov methods and applications to the fully coupled solution of the Navier-Stokes equations. *SIAM Review*, 48(4):700–721, 2006. doi: 10.1137/S0036144504443511

[23] R. S. Tuminaro, H. F. Walker, and J. N. Shadid. On backtracking failure in Newton-GMRES methods with a demonstration for the Navier-Stokes equations. *Journal of Computational Physics*, 180(2):549–558, 2002. doi: 10.1006/jcph.2002.7102.

[24] M. B. Giles. Accuracy of node-based solutions on irregular meshes. *11th International Conference on Numerical Methods in Fluid Dynamics*, 323(1):273–277, 1989. doi: 10.1007/3-540-51048-6_40.

[25] M. B. Giles. On adjoint equations for error analysis and optimal grid adaptation in CFD. *Oxford University Computing Laboratory, Technical Report*, 1997. url: http://eprints.maths.ox.ac.uk/1314/

[26] M. B. Giles, and N. A. Pierce. An introduction to the adjoint approach

to design. *Flow, Turbulence and Combustion*, 65(3-4):393–415, 2000. doi: 10.1023/A:1011430410075.

[27] M. B. Giles, and N. A. Pierce. Adjoint recovery of superconvergent functionals form PDE approximations. *SIAM Review*, 42(2):247–264, 2000. doi: 10.1137/S0036144598349423.

[28] M. B. Giles, M. Larson, M. Levenstam, and E. Suli. Adaptive error control for finite element approximations of the lift and drag coefficients in viscous flow. *Technical Report NA-76/06 Oxford University Computing Laboratory*, 1997. url: http://eprints.maths.ox.ac.uk/1317/

[29] R. Becker, and R. Rannacher. Weighted a posteriori error control in finite element methods. *Institut für Angewandte Mathematik*, Heidelberg, 1997.

[30] R. Becker, and R. Rannacher. An optimal control approach to a posteriori error estimation in finite element method. *Acta Numerica*, 10:1–102, 2001. doi: 10.1017/S0962492901000010.

[31] K. J. Fidkowski, and D. L. Darmofal. Review of output-based error estimation and mesh adaptation in computational fluid dynamics. *AIAA Journal*, 49(4):673–694, 2011. doi: 10.2514/1.J050073

[32] D. Darmofal, and D. Venditti. Anisotropic grid adaptation for functional outputs: application to two-dimensional viscous flows. *Journal of Computational Physics*, 187(1):22–46, 2003. doi: 10.1016/S0021-9991(03)00074-3.

[33] J.-D. Müller, and M. B. Giles. Solution adaptive mesh refinement using adjoint error analysis. In *Proceedings of the 15th AIAA Computational Fluid Dynamics Conference*, Anaheim, California, 2002.

[34] J. Ponsin, F. Fraysse, M. Gómez, and M. Cordero-Gracia. An adjoint-truncation error based approach for goal-oriented mesh adaptation. *Aerospace Science and Technology*, 41(1):229–240, 2015. doi: 10.1016/j.ast.2014.10.021

[35] M. Yano, J. M. Modisette, and D. L. Darmofal. The importance of mesh adaptation for higher-order discretizations of aerodynamic flows. In *Proceedings of the 20th AIAA Computational Fluid Dynamics Conference*, Honolulu, Hawaii, 2011.

[36] P. Roe. Approximate Riemann solvers, parameter vectors and difference schemes. *Journal of Computational Physics*, 43(2):357–372, 1981. doi: 10.1016/0021-9991(81)90128-5.

[37] P. Moinier, J.-D. Müller, and M. B. Giles. Edge-based multigrid and preconditioning for hybrid grids. *AIAA Journal*, 40(10):1954–1960, 2000. doi: 10.2514/2.1556

[38] R. P. Dwight. Heuristic a posteriori estimation of error due to dissipation in finite volume schemes and application to mesh adaptation. *Journal of Computational Physics*, 227(5):2845–2863, 2008. doi: 10.1016/j.jcp.2007.11.020

[39] F. Blom. Considerations on the spring analogy. *International Journal for Numerical Methods in Fluids*, 32(6):647–668, 2000. doi: 10.1002/(SICI)1097-0363(20000330)32:6<647::AID-FLD979>3.0.CO;2-K.

[40] L. Hascoët. Tapenade: a tool for automatic differentiation of programs. In *Proceedings of 4th European Congress on Computational Methods, ECCOMAS'2004*, Jyvaskyla, Finland, 2004.

[41] U. Ghia, S. Bayyuk, S. Habchi, C. .J. Roy, T. Shih, T. Conlisk, C. Hirsch, and J. M. Powers. The AIAA code verification project - test cases for CFD code verification. In *Proceedings of the 48th AIAA Aerospace Sciences Meeting Including the New Horizons Forum and Aerospace Exposition*, Orlando, Florida, 2010.

[42] T. Grätsch, and K.-J. Bathe. A posteriori error estimation techniques in practical finite element analysis. *Computers and Structures*, 83(4-5):235–265, 2005. doi: 10.1016/j.compstruc.2004.08.011

[43] A. Loseille, A. Dervieux, and F. Alauzet. Fully anisotropic goal-oriented

mesh adaptation for 3D steady Euler equations. *Journal of Computational Physics*, 229(8):2866–2897, 2010. doi: 10.1016/j.jcp.2009.12.021

[44] C. J. Roy. Review of discretization error estimators in scientific computing. In *Proceedings of the 48th AIAA Aerospace Sciences Meeting Including the New Horizons Forum and Aerospace Exposition*, Orlando, Florida, 2010.

[45] W. C. Tyson, J. M. Derlaga, C. J. Roy, and A. Choudhary. Comparison of r-adaptation techniques for 2-D CFD applications. In *Proceedings of the 22nd AIAA Computational Fluid Dynamics Conference*, Dallas, Texas, 2015.

[46] J.-D. Müller. Anisotropic adaptation and multigrid for hybrid grids. *International Journal for Numerical Methods in Fluids*, 40(3-4):445–455, 2002. doi: 10.1002/fld.313

[47] Z. Wang, K. Fidkowski, R. Abgrall, F. Bassi, D. Caraeni, A. Cary, H. Deconinck, R. Hartmann, K. Hillewaert, H. Huynh et al. High-order CFD methods: current status and perspective. *International Journal for Numerical Methods in Fluids*, 72(8):811–845, 2013. doi: 10.1002/fld.3767

[48] Y. Li, S. Premasuthan, and A. Jameson. Comparison of h-and p-adaptations for spectral difference methods. In *Proceedings of the 40th Fluid Dynamics Conference and Exhibit*, Chicago, Illinois, 2010.

[49] J. Majewski. An anisotropic adaptation for simulation of compressible flows. *Mathematical Modelling and Analysis*, 7(1):127–134, 2002. doi: 10.1080/13926292.2002.9637185

[50] M.-S. Liou. A sequel to AUSM, Part II: AUSM_{+}^{up} for all speeds. *Journal of Computational Physics*, 214(1):137–170, 2006. doi: 10.1016/j.jcp.2005.09.020

[51] V. Venkatakrishnan. Convergence to steady state solutions of the Euler equations on unstructured grids with limiters. *Journal of computational physics*, 118(1):120–130, 1995. doi: 10.1006/jcph.1995.1084

[52] V. Venkatakrishnan. On the accuracy of limiters and convergence to steady state solutionss. *31st Aerospace Sciences Meeting*, Reno, Nevada, 1993. doi: 10.2514/6.1993-880

[53] Z. Wang. A fast nested multi-grid viscous flow solver for adaptive Cartesian/Quad grids. *International Journal for Numerical Methods in Fluids*, 33(5):657–680, 2000. doi: 10.1002/1097-0363(20000715)33:5<657::AID-FLD24>3.0.CO;2-G.

[54] K. Michalak. and C. Ollivier-Gooch. Limiters for unstructured higher-order accurate solutions of the Euler equations. In *Proceedings of the 46th AIAA Aerospace Sciences Meeting and Exhibit*, Reno, Nevada, 2008.

[55] P. R. Spalart, and C. L. Rumsey. Effective inflow conditions for turbulence models in aerodynamic calculations. *AIAA journal*, 45(10):2544–2553, 2007. doi: 10.2514/1.29373.

[56] B. Wang, and G.-C. Zha. Comparison of a low diffusion E-CUSP and the Roe scheme for RANS calculation. In *Proceedings of the 46th AIAA Aerospace Sciences Meeting and Exhibit*, Reno, Nevada, 2008.

[57] A. Harten, P. D. Lax, and B. van Leer. On upstream differencing and Godunov-type schemes for hyperbolic conservation laws. *SIAM Review*, 25(1):35–61, 1983. doi: 10.1137/1025002

[58] P. Roe, H. Nishikawa, F. Ismail, and L. Scalabrin. On carbuncles and other excrescences. In *Proceedings of the 17th AIAA Computational Fluid Dynamics Conference*, Toronto, Ontario, 2005.

[59] H.-C. Lin. Dissipation additions to flux-difference splitting. *Journal of Computational Physics*, 117(1):20–27, 1995. doi: 10.1006/jcph.1995.1040

[60] M.-S. Liou. Mass flux schemes and connection to shock instability. *Journal of Computational Physics*, 160(2):623–648, 2000. doi: 10.1006/jcph.2000.6478

[61] M. Pandolfi, and D. 'Ambrosio. Numerical instabilities in upwind methods: analysis and cures for the carbuncle phenomenon. *Journal of Computational Physics*, 166(2):271–301, 2001. doi: 10.1006/jcph.2000.6652

[62] B. Diskin, J. L. Thomas, E. J. Nielsen, H. Nishikawa, and J. A. White. Comparison of node-centered and cell-centered unstructured finite-volume discretizations: viscous fluxes. *AIAA journal*, 49(7):1326–1338, 2010. doi: 10.2514/1.44940.

[63] B. Diskin. and J. L. Thomas. Effects of mesh regularity on accuracy of finite-volume schemes. In *Proceedings of the 50th AIAA Aerospace Sciences Meeting including the New Horizons Forum and Aerospace Exposition*, Nashville, Tennessee, 2012.

[64] E. Sozer, C. Brehm, and C. C. Kiris. Gradient calculation methods on arbitrary polyhedral unstructured meshes for cell-centered CFD solvers. In *Proceedings of the 52nd Aerospace Sciences Meeting*, National Harbor, Marylan, 2014.

[65] A. Jameson, W. Schmidt, E. Turkel. Numerical solutions of the Euler equations by finite volume methods using Runge-Kutta time-stepping schemes. In *Proceedings of the 14th Fluid and Plasma Dynamics Conference*, Palo Alto, California, 1981.

[66] A. Jameson. Time dependent calculations using multigrid, with applications to unsteady flows past airfoils and wings. In *Proceedings of the 10th Computational Fluid Dynamics Conference*, Honolulu, Hawaii, 1991.

[67] J. Nocedal, and S. J. Wright. *Numerical optimization.* Springer, New York, 2006.

[68] Y. Saad, and M. H. Schultz. GMRES: A generalized minimum residual algorithm for solving nonsymmetric linear systems. *SIAM Journal on Scientific and Statistical Computing*, 7(3):856–869, 1986. doi: 10.1137/0907058

[69] C. J. Roy. Review of code and solution verification procedures for com-

putational simulation. *Journal of Computational Physics*, 205(1):131–156, 2005. doi: 10.1016/j.jcp.2004.10.036

[70] P. J. Roache. Verification of codes and calculations. *AIAA journal*, 36(5):696–702, 1998. doi: 10.2514/2.457.

[71] J. L. Thomas, B. Diskin, and C. L. Rumsey. Towards verification of unstructured-grid solvers. *AIAA journal*, 46(12):3070–3079, 2008. doi: 10.2514/1.36655.

[72] Y. Bourgault, M. Picasso, F. Alauzet, and A. Loseille. On the use of anisotropic a posteriori error estimators for the adaptative solution of 3D inviscid compressible flows. *International Journal for Numerical Methods in Fluids*, 59(1):47–74, 2009. doi: 10.1002/fld.1797

[73] C. J. Roy. Strategies for driving mesh adaptation in CFD. In *Proceedings of the 47th AIAA Aerospace Sciences Meeting including The New Horizons Forum and Aerospace Exposition*, Orlando, Florida, 2009.

[74] M. Aftosmis, and M. Berger. Multilevel error estimation and adaptive h-refinement for cartesian meshes with embedded boundaries. In *Proceedings of the 40th AIAA Aerospace Sciences Meeting and Exhibit*, Reno, Nevada, 2002.

[75] M. Nemec, and M. Aftosmis. Aerodynamic shape optimization using a cartesian adjoint method and CAD geometry. In *Proceedings of the 24th AIAA Applied Aerodynamics Conference*, San Francisco, California, 2006. doi: 10.2514/6.2006-3456.

[76] P. E. Farrell, D. A. Ham, S. W. Funke, and M. E. Rognes. Automated derivation of the adjoint of high-level transient finite element programs. *SIAM Journal on Scientific Computing*, 35(4):369393, 2013. doi: 10.1137/120873558.

[77] J. Brown, and P. R. Brune. Low-rank quasi-Newton updates for robust Jacobian lagging in Newton methods. In *Proceedings of the 2013 Interna-*

tional Conference on Mathematics and Computational Methods Applied to Nuclear Science and Engineering, Sun Valley, Idaho, 2013.

[78] D. A. Knoll, and D. E. Keyes. Jacobian-free Newton-Krylov methods: a survey of approaches and applications. *Journal of Computational Physics*, 193(2):357–397, 2004. doi: 10.1016/j.jcp.2003.08.010.

[79] A. Choudhary, and C. J. Roy. A truncation error-based approach to understanding and improving mesh quality in CFD. In *Proceedings of the 50th AIAA Aerospace Sciences Meeting including the New Horizons Forum and Aerospace Exposition*, Nashville, Tennessee, 2012.

[80] K. J. Fidkowski, and P. L. Roe. An entropy adjoint approach to mesh refinement. *SIAM Journal on Scientific Computing*, 32(3):1261–1287, 2010. doi: 10.1137/090759057

[81] K. J. Fidkowski, M. A Ceze, and P. L. Roe. Entropy-based drag-error estimation and mesh adaptation in two dimensions. *Journal of Aircraft*, 49(5):1485–1496, 2012. doi: 10.2514/1.C031795

[82] D. A. Venditti, and D. L. Darmofal. Adjoint error estimation and grid adaptation for functional outputs: Application to quasi-one-dimensional flow. *Journal of Computational Physics*, 164(1):204–227, 2000. doi: 10.1006/jcph.2000.6600

[83] D. A. Venditti, and D. L. Darmofal. Grid adaptation for functional outputs: Application to two-dimensional inviscid flows. *Journal of Computational Physics*, 176(1):40–69, 2002. doi: 10.1006/jcph.2001.6967

[84] F. Fraysse, J. De Vicente, and E. Valero. The estimation of truncation error by τ-estimation revisited. *Journal of Computational Physics*, 231(9):3457–3482, 2012. doi: 10.1016/j.jcp.2011.09.031

[85] F. Fraysse, E. Valero, and J. Ponsin. Comparison of mesh adaptation using the adjoint methodology and truncation error estimates. *AIAA journal*, 50(9):1920–1932, 2012. doi: 10.2514/1.J051450.

[86] A. Demargne, R. Evans, P. Tiller, and W. N. Dawes. Practical and reliable mesh generation for complex, real-world geometries. In *Proceedings of the 52nd Aerospace Sciences Meeting, AIAA SciTech Forum*, National Harbor, Maryland, 2014.

[87] M. Gugala, M. Meyer, and J.-D. Müller. Towards an output-based remeshing for trubomachinery applications. In *Proceedings of the VII European Congress on Computational Methods in Applied Sciences and Engineering, ECCOMAS'2016*, pages 4011–4022, Crete Island, Greece, 2016.

[88] C. Dapogny, C. Dobrzynski, and P. Frey. Three-dimensional adaptive domain remeshing, implicit domain meshing, and applications to free and moving boundary problems. *Journal of Computational Physics*, 262:358–378, 2014. doi: 10.1016/j.jcp.2014.01.005.

[89] M. E. Hubbard. Multidimensional slope limiters for MUSCL-type finite volume schemes on unstructured grid. *Journal of Computational Physics*, 155(1):54–74, 1999. doi: 10.1006/jcph.1999.6329.

[90] T. J. Barth. Numerical aspects of computing high-Reynolds number flow on unstructuredmeshes. *29th AIAA Aerospace Sciences Meeting*, Reno, Nevada, 1991.

[91] T. J. Barth. *Aspects of unstructured grids and finite-volume solvers for the Euler and Navier-Stokes equations.* Von Karman Institute Lecture Series, Von Karman Institute for Fluid Dynamics, Belgium, 1994.

[92] W. L. Briggs. *A Multigrid Tutorial*. Siam, Philadelphia, 1987.

[93] J.-D. Müller. *On Triangles And Flow*. PhD thesis, The University of Michigan, 1996.

[94] S. Chapman. *Fortran 95/2003 For Scientists and Engineers*. McGraw-Hill Education, Boston, 2007.

[95] J. C. Adams, W. S. Brainerd, J. T. Martin, B. T. Smith, and J. L. Wagener. *Fortran 90 Handbook*, Volume 32. McGraw-Hill Education, New York, 1992.

[96] C. Geuzaine, and J.-F. Remacle. Gmsh, a three-dimensional finite element mesh generator with built-in pre- and post-processing facilities. *International Journal for Numerical Methods in Engineering*, 79(11):1309-1331, 2009. doi: 10.1002/nme.2579

[97] R. M. Stallman, R. McGrath, and P. D. Smith. GNU Make: A program for directing recompilation, for version 3.81. Free Software Foundation, 2004.

[98] R. P. Dwight. Robust mesh deformation using the linear elasticity equations. *Computational Fluid Dynamics 2006*, 401–406, Springer, Berlin, 2009. doi: 10.1007/978-3-540-92779-2_62

[99] L. Hascoët, and V. Pascual. The Tapenade Automatic Differentiation tool: Principles, Model, and Specification. *ACM Transactions on Mathematical Software, Association for Computing Machinery*, 39(3), Article No.20, 2013. doi: 10.1145/2450153.2450158

[100] V. Venkatakrishnan, and D. J. Mavriplia. *Implicit solvers for unstructured meshes*. ICASE, NASA Langley Research Center, Virginia, 1991.

[101] E. L. Allgower, and K. Georg. *Numerical continuation methods: An introduction*. Springer-Verlag, Berlin, 1990.

[102] J. Blazek. *Computational Fluid Dynamics: Principles and Applications*. Elsevier, Amsterdam, 2005.

[103] M. Bartholomew-Biggs. *Nonlinear optimization with engineering applications*. Springer-Verlag, Berlin, 2008.

[104] L. Y. Gicquel, G. Staffelbach, and T. Poinsot. Large eddy simulations of gaseous flames in gas turbine combustion chambers. *Progress in Energy and Combustion Science*38(6):782–817, 2012. doi: 10.1016/j.pecs.2012.04.004

[105] Ansys, Inc. Ansys meshing user's guide. *Vol*, 2013.

[106] V. Schmitt, and F. Charpin. Pressure distributions on the ONERA M6 wing at transonic mach numbers. *Experimental data base for computer program assessment*, 4(138):327–370, 1979.

[107] User Guide. Star-CCM+ version 6.04. 014. *Star-CCM*, 2011.

[108] Tutorials. Hyperworks 12.0. *Altair Engineering Inc*, 2013.

[109] J. Kortelainen. Meshing tools for open source cfd-a practical point of view. VTT, Espoo, Finland, 2009.

[110] M. Lanfrit. Best practice guidelines for handling automotive external aerodynamics with fluent. Fluent Deutschland GmbH, Darmstadt, 2005.

[111] G. Lloyd, and A. Espanoles. Best practice guidelines for marine applications of computational fluid dynamics. WS Atkins Consultants and Members of the NSC, MARNET-CFD Thernatic Network, London, 2002.

[112] J. Franke, A. Hellsten, H. Schlünzen, and B. Carissimo. *Best practice guideline for the CFD simulation of flows in the urban environment.* University of Hamburg, Meteorological Institute, Hamburg, 2007.

[113] R. Verfürth. A review of a posteriori error estimation and adaptive mesh-refinement techniques. *Computer Methods in Applied Mechanics and Engineering*176(1-4):419–440, 1999. doi: 10.1016/S0045-7825(98)00347-8

[114] T. M. Wildey, E. C. Cyr, R. P. Pawlowski, J. N. Shadid, and T. M. Smith. Adjoint based a posteriori error estimation in drekar:: CFD. Technical Report SAND2012-8910, Sandia National Laboratories, Springfield, 2012.

[115] F. Ismail, P. L. Roe, and H. Nishikawa. A proposed cure to the carbuncle phenomenon. *Computational Fluid Dynamics 2006*, 149–154, Springer, Berlin, 2009. doi: 10.1007/978-3-540-92779-2_21

[116] J.-R. Carlson. Inflow/outflow boundary conditions with application to FUN3D. NASA Center for AeroSpace Information, Hanover, 2011.

[117] S. R. Allmaras, and F. T. Johnson. Modifications and clarifications for the implementation of the Spalart-Allmaras turbulence model. *Seventh International Conference on Computational Fluid Dynamics (ICCFD7)*, 1–11, Hawaii, 2012.

[118] H. K. Versteeg, and W. Malalasekera. *An introduction to Computational Fluid Dynamics, The Finite Volume Method*. Pearson Education Limited, Glasgow, 1995.

[119] K. Peery, and S. Imlay. Blunt-body flow simulations. *24th Joint Propulsion Conference*, Boston, 1988. doi: 10.2514/6.1988-2904

[120] B. Diskin, and J. Thomas. Accuracy of gradient reconstruction on grids with high aspect ratio. *NIA Report* Number 2008-12, Hampton, 2008.

[121] T. Barth, and D. Jespersen. The design and application of upwind schemes on unstructured meshes. *AIAA Journal*, 366(13), 1989. doi: 10.2514/6.1989-366

[122] S. Xu. *CAD-based CFD shape optimisation using discrete adjoint solvers*. PhD thesis, Queen Mary University of London, London, 2015.

[123] R. Fedorenko. Relaxation method for solving differential elliptical equations. *USSR Computational Mathematics and Mathematical Physics*, 1(4):1092–1096, 1961. doi: 10.1016/0041-5553(62)90031-9

[124] N. Bakhvalov. On the convergence of a relaxation method with natural constraints on the elliptic operator. *USSR Computational Mathematics and Mathematical Physics*, 6(5):101–135, 1966. doi: 10.1016/0041-5553(66)90118-2

[125] A. Brandt, and O. E. Livne. *Multigrid techniques: 1984 guide with applications to fluid dynamics*, Volume 67. SIAM, USA, 2011. doi: 10.1137/1.9781611970753

[126] A. Brandt, and I. Yavneh. Improved coarse-grid correction for high Reynolds flows. In Multigrid methods IV, Hemker, P. and Wesseling, P., editors, volume 116 of *International Series of Numerical Mathematics*, 1–23. Birkhäuser Verlag, Basel (1994). Proceedings of the Fourth European Multigrid Conference, Amsterdam, July 6-9, 1993.

[127] A. Jameson. Solution of the Euler equations for two-dimensional transonic flow by a multigrid method. *Applied Mathematics and Computation*, 13(3-4):327–355, 1983. doi: 10.1016/0096-3003(83)90019-X

[128] D. Mavriplis, A. Jameson, and L. Martinelli. Multi-grid solution of the Navier-Stokes equations on triangular meshes. *AIAA Journal*, 28(8):1415–1425, 1990. doi: 10.2514/3.25233

[129] F. Christakopoulos. *Sensitivity computation and shape optimisation in aerodynamics using the adjoint methodology and Automatic Differentiation*. PhD thesis, Queen Mary University of London, London, 2013.

[130] B. Diskin, and J. L. Thomas. Accuracy analysis for mixed-element finite-volume discretization schemes. *National Institute of Aerospace Report*, Number 2007-08, 2007.

[131] D. R. Lindquist, and M. B. Giles. Comparison of numerical schemes on triangular and quadrilateral meshes. *1th International Conference on Numerical Methods in Fluid Dynamic*, 323:369–373, 1989. doi: 10.1007/3-540-51048-6_57.

[132] C. Othmer. A continuous adjoint formulation for the computation of topological and surface sensitivities of ducted flows. *International Journal for Numerical Methods in Engineering*, 58(8):861–877, 2008. doi: 10.1002/fld.1770.

[133] A. Jameson. Aerodynamic shape optimization using the adjoint method. *Lectures at the Von Karman Institute*, Brussels, 2003.

[134] A. Walther, and A. Griewank. Getting started with ADOL-C. *Department of Mathematics University of Paderborn*, Germany, 2009.

[135] A. I. Heft, T. Indinger, and N. A. Adams. Experimental and numerical investigation of the DrivAer model. *ASME 2012 Fluids Engineering Division Summer Meeting collocated with the ASME 2012 Heat Transfer Summer Conference and the ASME 2012 10th International Conference on*

Nanochannels, Microchannels, and Minichannels, 1(1):41–51, 2012. doi: 10.1115/FEDSM2012-72272

[136] A. Jaworski, J.-D. Müller, and J. Rokicki. One-shot optimisation with grid adaptation using adjoint sensitivities. *Proceedings ECCOMAS 2012*, 3685–3693, 2012.

[137] N. Forsythe. *A partitioned approach to fluid-structure interaction for artificial heart valves*. PhD thesis, School of Mechanical and Aero. Eng., Queen's University Belfast, Belfast, 2006.

[138] B. Van Leer, W.-T. Lee, and P. Roe. Characteristic time-stepping or local preconditioning of the Euler equations. *10th AIAA Computational Fluid Dynamics Conference*, Hawaii, 1991. doi: 10.2514/6.1991-1552

[139] B. Van Leer, and C.-H. Tai. Design of optimally smoothing multi-stage schemes for the Euler equations. *9th Computational Fluid Dynamics Conference*, Buffalo, NY, 1989.

[140] C.-H. Tai, J.-H. Sheu, and B. Van Leer. Optimal multistage schemes for Euler equations with residual smoothing. *AIAA Journal*, 33(6):1008–1016, 1995.

[141] F. Fraysse. *Numerical Error Prediction and its Applications in CFD using tau-estimation*. PhD thesis, E.T.S.I. Aeronauticos Universidad Politcnica de Madrid, Madrid, 2012.

[142] J.-D. Müller, T. Schonfeld, and M. Rudgyard. A comparison of the treatment of hanging nodes for hybrid grid refinement. *13th Computational Fluid Dynamics Conference*, Snowmass Village, 1997. doi: 10.2514/6.1997-1859

[143] T. Phillips. Residual-based discretization error estimation for computational fluid dynamics. *Dissertation submitted to the Faculty of the Virginia Polytechnic Institute and State University in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Aerospace Engineering*, Blacksburg, 2014.

[144] P. R. Spalart, and S. R. Allmaras. A one-equation turbulence model for aerodynamic flows. *La Recherche Aerospatiale*, 1(1):5–21, 1994. doi: 10.2514/6.1992-439

[145] M. Desbrun, M. Meyer, P. Schröder, and A. H. Barr. Implicit fairing of irregular meshes using diffusion and curvature flow. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, 317–324, ACM Press/Addison-Wesley Publishing, New York, 1999.

[146] L. Armijo. Minimization of functions having continuous partial derivatives. *Pacific Journal of Mathematics*, 16(1):1–3, 1966. doi: 10.2140/pjm.1966.16.1.